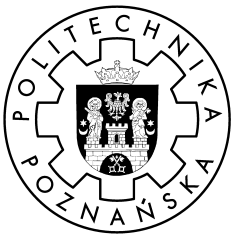
Poznan University of Technology
Faculty of Computing Science
Institute of Computing Science



Doctoral dissertation

HEURISTIC ALGORITHMS FOR DISCOVERY OF SEARCH OBJECTIVES IN TEST-BASED PROBLEMS

Paweł Liskowski

Supervisor
Krzysztof Krawiec, Ph. D., Dr. Habil., Professor

Poznań, 2018

To my parents,
from whom it all began

Acknowledgments

The work described here was carried out between October 2013 and May 2018 in the Laboratory of Intelligent Decision Support Systems at the Faculty of Computing at Poznan University of Technology. I would like to thank all people who has supported me and who kept me going throughout the effort of completing this thesis.

First and foremost, I would like to express my deep gratitude to my supervisor Krzysztof Krawiec, who took me under his wings and introduced me to the thrilling world of machine learning. I deeply thank him for the enormous effort he put in my guidance over the past years. His enthusiasm, inspiration and encouragement were all invaluable to me. He has always been there for me, and showed me a lot of heart on every occasion. Our collaboration has affected my life in ways that I cannot begin to explain. For this, and so much more, I am forever grateful.

I am also thankful for the advice and support of my dear friend Wojciech Jaśkowski, who has not only shown a large interest in my work, but also guided me at the very beginning of my academic journey. His expertise, useful criticism and countless conversations have helped me achieve many of my goals.

I would like to thank my friends and collaborators: Marcin Szubert, Tomasz Pawlak, Bartosz Wieloch, Iwo Błądek, Thomas Helmuth, William La Cava and Lee Spector for many insightful discussions and great moments we shared together.

Furthermore, I would like to immensely thank my parents, Jolanta and Marian, my sister Daria, and my wife Malwina whose unconditional support, love and understanding gave me strength and motivation to pursue my goals.

Finally, I acknowledge the support of the Polish National Science Centre grant no. DEC-2014/15/N/ST6/04572.

Abstract

A wide spectrum of optimization and machine learning problems approached in evolutionary computation involve evaluation functions that reward candidate solutions by counting the number of tests they pass. The common features of these problems, known in computational intelligence as test-based problems, are that the number of tests may be large (or even infinite), interactions with them are largely independent, and each of them produces an outcome that may provide the agent with useful feedback.

Conventionally, the discrepancy between the actual and the desired outcome of confronting solutions with tests are aggregated into a scalar evaluation function. In this thesis, we demonstrate that the habit of driving search using such function leads to evaluation bottleneck, i.e. information-rich process of candidate solutions' evaluation is compressed into a scalar value, which necessarily incurs information loss and so cripples the performance of evolutionary algorithms. We investigate the possibility of broadening the bottleneck of scalar evaluation by making evolutionary search algorithms better-informed about the outcomes of interactions between candidate solutions and tests. To this end, we propose a family of methods aimed at automatic discovery of heuristic search objectives that provide multi-criteria evaluation of candidate solutions. The key concept behind the algorithms that implement discovery of search objectives is an interaction matrix, which holds detailed account on interaction outcomes with individual tests. Crucially, the search objectives reflect only selected characteristics of candidate solutions. Rather than providing objective assessment of candidate solution's quality, their role is to guide search by creating a useful gradient towards better performing solutions.

We propose three different algorithms that implement these ideas. The first of them, DOC, automatically identifies the groups of tests for which the candidate solutions behave similarly. Each such group gives rise to a new search objective. The second method, DOF, learns the factors that jointly model an interaction function and uses them as search objectives. Finally, we propose SFIMX, a method dedicated primarily to reducing the computational complexity of evaluation in test-based problems, which computes only a fraction of interactions between candidate solutions and tests, and then predicts the outcomes of the remaining ones.

We gather these methods under the common conceptual framework of heuristic algorithms for discovery of search objectives in test-based problems. When applied to several well-known test-based problems, including the game of Othello, the Iterated Prisoner's Dilemma, Density Classification Task, as well as a suite of program synthesis tasks, the proposed algorithms significantly outperform the conventional search algorithms driven by scalar evaluation.

The results obtained in this dissertation open the door to efficient and generic countermeasures to premature convergence for both coevolutionary and evolutionary algorithms applied to problems featuring aggregating fitness functions.

Preface

Some ideas, figures and portions of text presented in this dissertation have appeared previously in the following publications:

[1] Wojciech Jaśkowski, Paweł Liskowski, Marcin Szubert, and Krzysztof Krawiec. Improving Coevolution by Random Sampling. In *Proceedings of the 15th annual conference on Genetic and Evolutionary Computation*, pages 1141–1148. ACM, 2013.

[2] Paweł Liskowski and Krzysztof Krawiec. Discovery of Implicit Objectives by Compression of Interaction Matrix in Test-based Problems. In *Parallel Problem Solving from Nature–PPSN XIII*, pages 611–620. Springer International Publishing, 2014.

[3] Krzysztof Krawiec and Paweł Liskowski. Automatic Derivation of Search Objectives for Test-based Genetic Programming. In Penousal Machado, Malcolm I. Heywood, James McDermott, Mauro Castelli, Pablo Garcia-Sanchez, Paolo Burelli, Sebastian Risi, and Kevin Sim, editors, *18th European Conference on Genetic Programming*, volume 9025 of *LNCS*, pages 53–65, Copenhagen, 2015. Springer.

[4] Paweł Liskowski, Krzysztof Krawiec, Thomas Helmuth, and Lee Spector. Comparison of Semantic-aware Selection Methods in Genetic Programming. In *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*, GECCO Companion '15, pages 1301–1307, New York, NY, USA, 2015. ACM.

[5] Wojciech Jaśkowski, Paweł Liskowski, Marcin Szubert, and Krzysztof Krawiec. Performance Profile: A Multi-criteria Performance Evaluation Method for Test-based Problems. *International Journal of Applied Mathematics and Computer Science*, 26(1):215–229, 2016.

[6] Paweł Liskowski and Krzysztof Krawiec. Non-negative Matrix Factorization for Unsupervised Derivation of Search Objectives in Genetic Programming. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, GECCO '16, pages 749–756, New York, NY, USA, 2016. ACM.

[7] Paweł Liskowski and Krzysztof Krawiec. Online Discovery of Search Objectives for Test-based Problems. *Evolutionary Computation*, 25(3):375–406, 2016.

[8] Paweł Liskowski and Krzysztof Krawiec. Surrogate Fitness Via Factorization of Interaction Matrix (Best-paper Award winner). In Malcolm I. Heywood, James McDermott, Mauro Castelli, and Ernesto Costa, editors, *EuroGP 2016: Proceedings of the 19th European Conference on Genetic Programming*, volume 9594 of *LNCS*, pages 65–79, Porto, Portugal, 30 March–1 April 2016. Springer Verlag.

[9] Paweł Liskowski and Wojciech Jaśkowski. Accelerating Coevolution with Adaptive Matrix Factorization, (nominated to Best-paper Award). In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 457–464, New York, NY, USA, 2017. ACM.

[10] Paweł Liskowski and Krzysztof Krawiec. Adaptive Test Selection for Factorization-based Surrogate Fitness in Genetic Programming. *Foundations of Computing and Decision Sciences*, 42(4):339–358, 2017.

[11] Paweł Liskowski and Krzysztof Krawiec. Discovery of Search Objectives in Continuous Domains. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 969–976, New York, NY, USA, 2017. ACM.

Contents

Chapter 1

Introduction

1.1 Problem Setting and Motivation

In his groundbreaking work, Alan Turing defined the notion of computation by an analogy to a human mathematician who carefully performs elementary calculations, moving step-by-step towards the exact solution [346]. Even though Turing's vision is now largely fulfilled, i.e., modern computers excel at simple arithmetic and are easily *programmable*, it is tasks like recognizing objects in images, winning a game of Go, or automatic programming — problems where the rules are not clear, some of the necessary information is missing, or finding the right answer/solution would require considering an exorbitant number of possibilities — that now pose the biggest challenges in computer science. The algorithms the researchers have developed to solve many of the hardest classes of problems have moved emphasis away from exact and robust computing. Instead, tackling real-world tasks often involves heuristics and difficult compromises (e.g. trading off precision with time), but most importantly, it entails *intelligence*.

Ever since the inception of programmable computers, humans wondered whether machines might become intelligent, over a hundred years before one was actually built [224]. The dream of creating machines that think dates back to at least the time of ancient Greece; intelligent constructs appear in literature since then [266]. Nowadays, artificial intelligence (AI) is a thriving field with great prospects, rich community of researchers and many practical applications. The undeniable sign of the latter is that we already let intelligent software automate production, process speech and images, or even detect faults in machinery and perform a medical diagnosis. Though it may be hard to embrace, machines have already achieved better results than humans on some of these tasks [326, 85, 343, 125, 215].

This thesis concerns problems which emerged in computational intelligence (CI) [105], a branch of AI dedicated to problem-solving by means of bio-inspired algorithms. Though biological roots are undoubtedly very important, neural, evolutionary and related approaches are only the tip of the iceberg when it comes to what actually CI community works on. According to Włodzisław Duch "*Computational intelligence is a branch of computer science studying problems for which there are no effective computational algorithms.*" [78]. Indeed, shifting the focus from algorithms to problems is an important change of perspective, as the true challenge of AI proved to be solving the tasks that are considered easy for humans, but hard to describe formally, i.e. problems that humans solve intuitively, like understanding of speech or recognizing objects in a scene. In many cases, the most convenient (and sometimes the only possible) way to express the *intent* when approaching such tasks is to provide a set of *examples* that characterize the desired behavior, or describe the properties of an ideal solution.

Put another way, CI abounds in *test-based problems*, i.e. problems that feature *tests* (examples), entities that embody pieces of knowledge about the problem and are considered to be actual instances representing a given task. Training examples in machine learning, opponents in games, and environments in reinforcement learning and robotics are all examples of tests. In these settings, an agent interacts with tests and learns from the outcomes of those interactions: a machine learning inducer builds or corrects a hypothesis, a game-playing algorithm adjusts its strategy, and a virtual or physical robot updates its policy. The common features of these scenarios are that the number of tests may be large (or even infinite) and interactions with each of them are largely independent.

Test-based problems attracted much attention in *coevolutionary algorithms*, a branch of evolutionary computation. *Competitive coevolution* devised there assumes iterative co-adaptation of *candidate solutions* (entities intended to solve a given problem) with *tests*. In the simplest case, the candidate solutions and tests dwell in separate populations, and the former are rewarded for the number of passed tests, while the latter for the number of solutions they fail.

The conceptual framework of test-based problems comes in handy also in the more general setting of an evolutionary algorithm, where the set of tests is fixed and given as a part of problem formulation, like the training set of examples in machine learning. This is the default setting for *genetic programming* that is also in the focus of this thesis, where candidate solutions are symbolically represented executable structures like programs or expressions.

The challenge in designing effective algorithms for test-based problems lies, among others, in obtaining accurate evaluation of candidate solutions. An objective assessment of solution's performance is often computationally infeasible, precluding its application in practice. For instance, arguably the most popular evaluation function for many test-based problems is the *expected utility,* defined as the the average interaction outcome against *all* tests. Even for the apparently simple problem of learning a strategy for the game of Tic-Tac-Toe, computing the expected utility requires playing games against staggering 3.47×10^{162} unique opponent strategies [145]. The conventional approach to reduce the computational complexity of evaluation in test-based problems is to limit the number of tests. In case of expected utility, this means computing a limited number of interactions between solutions and tests. Crucially, outcomes of those interaction are typically aggregated into a single scalar value, which is then used to drive a search algorithm.

An evaluation function that counts the number of passed tests, such as the expected utility, usually forms an inherent part of the problem and makes it amenable to many conventional search algorithms that expect a scalar objective. Though convenient and compact, aggregation of interaction outcomes inevitably leads to information loss, primarily due to *compensation*: two solutions that pass k tests each are considered equally valuable, no matter *which* particular tests they pass. Also, aggregation neglects the fact that some tests can be inherently more *difficult* than others, not to mention that it is sometimes the capability to solve a *specific combination* of tests (rather than a *number* of tests) that matters the most for further progress. Algorithms that rely on aggregation are oblivious to these aspects and thus prone to inferior performance.

In this thesis, we identify and address the *evaluation bottleneck* problem that arises when search algorithms are driven using scalar evaluation [194]. As demonstrated above, candidate solutions are often very complex entities (game-playing agents, computer programs, etc.), yet all that remains from the process of their evaluation is merely a scalar value. The habit of using scalar evaluation (though fully justified in rare scenarios) appears to be deeply rooted within the CI community, even though there are no principal reasons for an outcome of evaluation to be that succinct. This is particularly true for test-based problems, where an act of evaluating a candidate solution involves interaction with *multiple* tests and produces detailed information that can be potentially exploited

to benefit the search. Our main motivation is therefore to widen the evaluation bottleneck by providing search algorithms with richer information on solutions' characteristics.

1.2 Aims and Scope

Following the above discussion, in this thesis our main goal is to broaden the bottleneck of scalar evaluation by making evolutionary search algorithms better-informed about per-test effects of computation, i.e., outcomes of interactions between candidate solutions and tests. Let us emphasize that the focus is here not on the search algorithm *per se*, but on the algorithms that elicit detailed information on behavior of candidate solutions. Their role is to provide an interface between evaluation function and the other components of search algorithm (e.g. selection operator) that eases the communication and minimizes loss of information.

As argued in the previous section, evaluation in test-based problems has the potential of providing extensive information on behavior of candidate solutions. With this in mind, our secondary objective is to harness this information for the purpose of providing a useful gradient for a search process. To achieve this goal we propose a novel family of methods aimed at *automatic discovery of multi-aspect characterizations of candidate solutions*, gathered under the common conceptual umbrella of *search objectives*.

Moreover, in accordance with the prevailing trend in AI, we avoid explicit incorporation of domain knowledge into discovery of search objectives. Rather than that, we design the process to be automatic, largely data-driven and knowledge-free [234]. To this end, we employ *learning* via unsupervised learning algorithms to capture the underlying patterns in behavior of candidate solutions on tests.

The specific objectives of this thesis are as follows:

1. To investigate the phenomenon of evaluation bottleneck and possible ways of dealing with its consequences in the context of algorithms solving test-based problems.
2. To propose a measure of test difficulty and devise the concept of performance profile, a multi-criteria evaluation method that characterizes candidate solutions using a vector of outcomes against tests of various difficulty.
3. To demonstrate the practical utility of performance profiles for comparing candidate solutions obtained with various learning algorithms for test-based problems.
4. To define the concept of search objective, a formal object designed to encapsulate the problem objectives and guide a search process by creating a useful gradient toward better quality solutions.
5. To formalize the framework for automatic discovery of search objectives, which serves the purpose of broadening the evaluation bottleneck and acquiring alternative (or additional) behavioral information from candidate solutions and unifies approaches operating on interaction matrices.
6. To develop algorithms for automatic discovery of search objectives that provide search algorithms with richer information on solutions' behavior and demonstrate how such characteristics can be embedded in a search algorithm to facilitate a better-informed and directed search.
7. To experimentally verify the proposed algorithms on selected test-based problems and compare them to the reference methods driven by scalar evaluation.

1.3 Thesis Outline

This dissertation is organized as follows.

Chapter 2 provides a brief overview of the field of evolutionary computation. We introduce the basic evolutionary algorithm and characterize its main components. Next, we discuss the key principles of evolutionary search, with the focus on the role of fitness function. We summarize the chapter by sketching the main problem we tackle in this thesis, i.e. the difficulty of choosing the right evaluation function.

Chapter 3 presents the mathematical model of natural coevolution, with the emphasis on so-called competitive coevolution. We delineate different variants of the algorithm, including one- and multi-population coevolution, and discuss the most important differences between coevolutionary and evolutionary approach to problem-solving. Afterwards, we provide a brief literature review of existing models of coevolution and their practical applications. Finally, we discuss coevolutionary pathologies that hinder the overall progress of such methods.

Chapter 4 describes the framework for genetic programming, one of the earliest paradigms for automatic programming and evolution of computer programs. We describe the canonical variant of the algorithm and provide the overview of its key underlying components. We also discuss the contemporary challenges in genetic programming and present some of its most impressive human-competitive achievements.

Chapter 5 introduces the class of test-based problems that proves useful when modeling domains with no intrinsic objective measure. We provide a formal definition of a test-based problem, oriented at the concept of interaction between a candidate solution and a test. On this basis, we describe how (co)evolutionary algorithms can be applied to such problems. This chapter also presents some of the experimental domains used throughout this thesis to validate the methods proposed in subsequent chapters.

Chapter 6 identifies the pitfalls of scalar evaluation that originate in the aggregation of outcomes of multiple interactions between a candidate solution and a set of tests. In particular, we identify the problem of evaluation bottleneck and discuss its implications for search algorithms.

Chapter 7 proposes performance profiles, a means for many-aspect assessment of solutions produced by algorithms applied to test-based problems. We devise two different approaches to building such profiles and validate them experimentally in the game of Othello and the Iterated Prisoner's Dilemma. We also demonstrate how a performance profile exposes differences in behavior of candidate solutions that would pass unnoticed by a scalar evaluation.

Chapter 8 formalizes the framework for automatic discovery of search objectives in test-based problems, intended to widen the evaluation bottleneck by providing search algorithms with richer information on solutions' characteristics. We describe its conceptual underpinnings, including the notions of interaction matrix and derived search objective. On this basis, we demonstrate how the framework facilitates automatic design of heuristic evaluation functions and discuss several possible ways in which they can be employed as alternative means of driving a search algorithm.

Chapter 9 proposes an algorithm for discovery of search objectives by heuristic clustering of outcomes of interactions taking place between candidate solutions and tests. We demonstrate that the method manages to produce a low number of objectives that approximately capture the capabilities of evolving solutions. We analyze its properties, proving that the discovery process preserves the dominance relation between candidate solutions in the space of search objectives. When applied to several well-known test-based problems, the proposed approach significantly outperforms the conventional (co)evolutionary search algorithms driven by scalar evaluation.

Chapter 10 introduces an algorithm for discovery of search objectives by factorization. The proposed approach relies on the machine learning technique of non-negative matrix factorization to learn latent factors that characterize candidate solutions and tests. These factors are the primary building blocks of search objectives derived by the method. We demonstrate that, when employed to drive search, they foster diversification of search directions, while maintaining a useful search gradient for the entire evolution. The approach is validated experimentally on a range of program synthesis tasks, where it proves to achieve higher success rates than discovery of search objectives by clustering.

Chapter 11 extends the algorithms presented in the previous two chapters, that are originally limited in being applicable only to problems with constrained interaction outcomes, to domains where interaction outcomes are continuous. We perform an extended experimental evaluation of the methods on a range of uni- and multivariate symbolic regression benchmarks. Additionally, we hybridize our approach with lexicase selection and thoroughly evaluate the resulting configurations.

Chapter 12 uses the framework for discovery of search objectives to devise a surrogate fitness technique aimed at reducing the overall computational cost of evaluation in test-based problems. The proposed method calculates a partial interaction matrix between candidate solutions and tests, and then employs non-negative matrix factorization to predict its missing entries. We first demonstrate the effectiveness of the method in multiple scenarios and then develop several extensions that further improve its performance, also rendering the method virtually parameter-free.

Chapter 13 summarizes the dissertation, reviews the main contributions and outlines the promising directions for future work.

Chapter 2

Evolutionary Computation

Evolutionary computation (EC) is an area of research deeply embedded within computing science. Contemporarily, most consider it is a subfield of artificial intelligence or, more specifically, computational intelligence [82]. Natural processes have always served as a source of inspiration for the human kind, thus it is not surprising that the scientists pursued the idea of adopting biological principles into computation. The use of Darwinian principles for automated problem solving dates back to the 1950s. A few years later, in the early 1960s, three independent framings of these ideas started to emerge, giving birth to the mainstream of evolutionary computation [99, 134, 296]. In the last few decades, the continuous advancement of technology encouraged the application of EC to virtually every area of problem solving. As impressive as it may seem, evolutionary computation paradigm has proven to be an immensely powerful problem-solving strategy, demonstrating the applicability of evolutionary principles. EC has been successfully used in a wide variety of fields to evolve solutions to difficult problems such as cancer detection [97, 98], automatic evolution of computer programs [176], modeling of adaptive systems by means of genetic programming [134], complex engineering problems [296], or learning in games [310, 91, 93]. Most importantly, as larger and more difficult problems are considered, the solutions discovered by evolutionary methods are often more efficient and robust than those designed explicitly by humans.

2.1 Evolutionary algorithms

For the past few decades, many different variants of EC paradigm have been proposed, including Holland's genetic algorithms (GAs) [134], Rechenberg's and Schwefel's evolution strategies (ES) for parameter optimization problems [296, 319] or Koza's genetic programming (GP) [176], to name just the most popular ones. All these computational models are bio-inspired optimization procedures that involve reproduction, random variation, and selection of contending individuals in a population. Jointly, these features are essential for evolution, and once all of them are combined together, whether in nature or in a artificial simulation, evolution is inevitable [10]. From yet another perspective, they are also variants of *metaheuristics*, i.e., a general-purpose search strategy that is applicable to a wide variety of search and learning tasks.

These and other flavors of EC rely heavily on a common prototype in the form of evolutionary algorithm (EA). Evolutionary algorithms mimic the process of natural evolution, and in contrast to the classic optimization techniques of gradient descent, dynamic programming, branch and bound or local search, EAs are *population-based*. Instead of processing just a single search point at a time, EAs maintain a *population of individuals* that allows them to sample and simultaneously process many points in the space of potential solutions. By employing a number of individuals,

EAs perform an efficient directed search and are less susceptible to stagnation at local optima. Usually, initial population is filled with random solutions (elements of the search space).

When talking about EAs, it is common to borrow some vocabulary from genetics. We refer to candidate solutions in a population as *individuals* or *genotypes* (sometimes also chromosomes). The interpretation of each genotype, i.e., its *phenotype* is typically problem-dependent and defined externally. Each individual is made of constituents called *genes* (features). For instance in GAs, genes are arranged in linear succession and every gene controls inheritance of certain characteristic. In a given individual, a gene is in one of several states, called *alleles* (values). When run on a population of genotypes, an evolutionary process corresponds to a search through a space of potential solutions.

Central to all EAs is the idea of searching the problem space by changing an initially random population of individuals in such a manner that the fitter (better) individuals have more chances to survive and pass their genes (information) to subsequent generations. This *survival of the fittest* is used to refine a population of individuals by applying a *selection* procedure that favors better individuals and causes them to reproduce more often. The quality of individuals is typically measured using an evaluation function called *fitness*. EAs are generally applied to solve optimization problems (cf. Chapter 5), i.e., finding a solution that maximizes or minimizes a given *objective function*. In such a case, the evaluation function (fitness function) is often identical to the given objective function. In most applications, evaluation is the only component that has to be specifically tailored to the problem at hand, while the remaining components are usually generic and elaborated in the theory and practice of EC (e.g., bit-string representation, two-point crossover, tournament selection, etc.).

Another critical component of EAs are operators that are inspired by the natural phenomena of *mutation* and *recombination* that provide for *variation*, one of the cornerstones of evolution. Their purpose is to create the offspring and introduce both diversity and novelty into a new population. The main characteristic of a mutation operator is that it is being applied to a single individual to produce a new individual. Most mutation operators are also designed in such a way that, for some metric defined in the space of candidate solutions, they produce an offspring that is relatively close to the parent solution. In some cases, this closeness is controlled by the *strength* of mutation. For instance, in the canonical Gaussian mutation, the parameter σ determines the magnitude of normally distributed random noise applied to a single variable (gene) in the candidate solution (genotype).

Recombination (or *crossover*) operators create new individuals by combining parts from two (or occasionally more) parents, so that the offspring inherits some of their traits. For instance in tree-based GP (cf. Section 4.2), a crossover operator swaps two randomly selected subtrees in parents' trees.

A typical EA workflow engages both mutation and recombination, because they complement each other. Apart from generating new points in a search space, mutation also maintains so-called 'gene pool' — the set of unique alleles available to recombination in the population. Most recombination operators produce new individuals by exchanging parts of genotypes of parents' solutions. If the population is not diverse enough, the opportunity for recombination operators to perform useful search diminishes rapidly.

All of the introduced above components of an EA are usually implemented as stochastic algorithms. For instance, during selection the fitter individuals are more likely to be selected than the less fit ones, but even the worst individuals in the population may become parents to the solutions in the next generation. For recombination, parts of the genotype that are exchanged between the

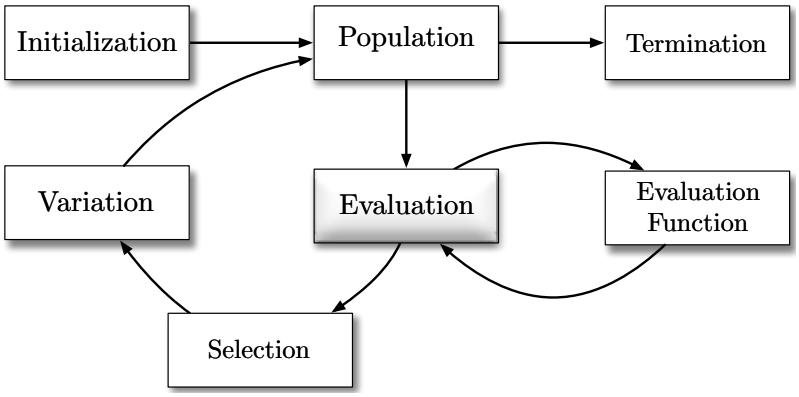


Figure 2.1: General scheme of an evolutionary algorithm presented as a flow-chart.

parents are chosen randomly. Similarly for mutation, the choice regarding which pieces to mutate and which new genes to replace them with is random.

Evolutionary algorithms mimic the process of natural evolution that can be viewed as the result of interplay between the process of creation of new individuals, their evaluation and selection. This neo-Darwinian model of evolution is reflected by the structure of a typical generational EA as illustrated in Fig. 2.1. The initial population of individuals is created in a randomized way and then evaluated using a fitness function. Afterwards, the algorithm proceeds iteratively in generations until the termination condition is met. In each generation a new population is formed by selecting the fittest individuals to drive the search process towards better solutions. Some members of the new population are then chosen to act as parents for the new individuals and undergo transformations by means of genetic variation operators (typically mutation and crossover). Finally, the newly-bred individuals are evaluated and the cycle repeats. Over time, the evolutionary loop leads to an increased presence of highly-fit individuals in the population, while the non-deterministic nature of variation operators ensures constant flow of 'fresh blood' in the form of novel offspring. The termination condition typically depends on the number of elapsed generations. However, it may also depend on the quality of the best solution found so far, or any other criteria appropriate for the problem at hand. When the termination condition is fulfilled, the evolution stops and the fittest individual is returned as the final solution.

It is rather easy to notice that the scheme of an evolutionary algorithm falls into the category of *generate-and-test* algorithms. In each generation, variation operators generate new candidate solutions, and a fitness function is used to test their quality. Generate-and-test approach is the essential difference that distinguishes EAs from, among others, gradient-based algorithms, where new search points are obtained from the current ones via directed updates dictated by a gradient.

As with any other technique, EAs come in many flavors and twists. The above overview gives only a general idea of how a typical EA proceeds, without delving into details of how each phase can be implemented. For instance, there are many selection methods, including tournament selection, where a random sample of individuals compete for survival and reproduction, fitness proportional selection [15], where the probability of selection is proportional to individual's fitness, or even more elaborate multi-objective selection techniques such as NSGA-II [75]. Yet, these examples are just the tip of the iceberg in a plethora of available methods, not just when it comes to selection. Vast body of literature and past research is devoted to disseminating initialization, recombination and evaluation techniques that are dedicated to particular problem classes and applications. The extensive presentation of evolutionary algorithms can be found in the book of Eiben and Smith [80], while their history and applications are surveyed by Bäck et al. [16].

2.2 Evolutionary search

Evolutionary computation draws inspiration from natural evolving systems to build problem-solving algorithms. The range of problems amenable to solution via evolutionary methods includes optimization, constraint satisfaction, as well as more general forms of adaptation, but virtually all these are *search* problems. It is therefore reasonable to begin by introducing the basic terminology of search and review the key ideas underpinning its analysis.

The goal of a search problem is usually to find one or more candidate solutions satisfying some predefined properties. These properties are usually defined with respect to an *objective function*, which generally takes the form

$$f_o : \mathcal{S} \to \mathbb{R}, \tag{2.2.1}$$

where $\mathbb{R}$ is the set of real numbers. A *search space* is the domain of the function f_o, or in other words, a set of feasible points that might be considered during search. Depending on the problem it might be finite or infinite, continuous or discrete. In the above case, the search space would typically be $\mathcal{S}$, or possibly some subset of $\mathcal{S}$. To give a more concrete example, if the problem is to synthesize a program that produces the value of the Boolean even parity given n independent Boolean inputs, the search space could be chosen to be the set of expression built of Boolean functions such as `and`, `or`, `nand`, `nor` accompanied by binary terminals 0 and 1 (cf. Section 4.2). In any case, points in the search space $\mathcal{S}$ are typically referred to as *solutions* or *candidate solutions*.

In the special case of search problem being an optimization problem, which is most often the case in EC, the goal is to find one or more points in the search space which maximize or minimize f_o. Naturally, since $\max f_o = -\min(-f_o)$, both approaches are equivalent.

Let us now assume that our search problem is an optimization problem and, without loss of generality, that the objective is to minimize f_o. Then the *global optima* are precisely those points in $\mathcal{S}$ for which f_o is a minimum. More formally, we search for a solution such that

$$x^* \in \mathcal{S} \iff f_o(x^*) = \min_{s \in \mathcal{S}} f_o(x),$$

or equivalently:

$$x^* = \arg\min_{x \in \mathcal{S}} f_o(x).$$

Of course, when f_o is differentiable other techniques than EAs can be employed. In many practical situations, the natural choice is the gradient descent method (also known as steepest descent or Cauchy's method). Assuming that f_o is a twice differentiable function, Newton-Raphson based on the second order Taylor series expansion can also be used. For some problems, the solution can be even obtained analytically by determining the zeros of the gradient and verifying positive definiteness of the Hessian matrix at candidate points. If f_o is a convex function, i.e it has only one (global) optimum, then the problem can be solved by convex optimization methods like for instance subgradient projection [29].

However, deterministic optimization methods may only converge to a *local optima* in case of multi-modal problems. Also, derivatives, gradients and other concepts mentioned above are not available in discrete search spaces that are often of practical interest (and studied in this thesis). Before discussing local optima, let us first introduce the notion of the *neighborhood* of point x. Given a metric d in solultion space, $d : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, the neighborhood of point $x \in \mathcal{S}$ is the set of points within ϵ distance of x with respect to the metric d:

$$N(x, \epsilon) = \{y \in \mathcal{S} | d(x, y) \leq \epsilon\}.$$

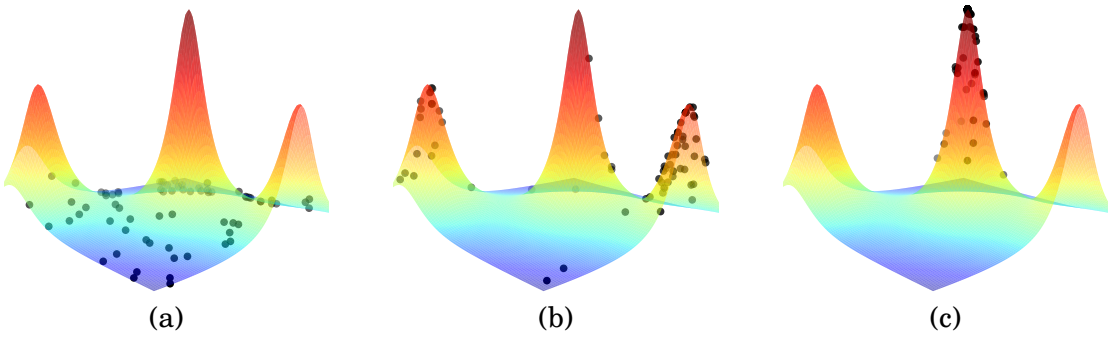


Figure 2.2: A fitness landscape of two-dimensional objective function, which shows how fitness depends on individual's phenotype. Subsequent figures illustrate the progress made by EA on this landscape with population of 100 individuals over the course of 10 generations. Black dots correspond to individuals and show their distribution at the beginning (a), in the middle (b), and at the end of evolutionary run (c).

Local optimum refers to a solution which is optimal within a *neighboring* set of candidate solutions. More formally, a local optimum is a point for which, for small enough ϵ, no member of its ϵ-neighborhood has a lower objective function value, so that:

$$\mathcal{L} = \{x \in \mathcal{S} | \exists\, \epsilon > 0 \wedge \forall\, y \in N(x, \epsilon) : f(x) \leq f(y)\}.$$

In order to gain a deeper understanding of search processes, it is sometimes convenient to imagine f_o as defining a *fitness landscape* [366], or the surface spanned over $\mathcal{S}$, where a solution corresponds to a point on the landscape, and the elevation of that point represents its objective function value. In such a setting, traditional gradient descent methods can be likened to a ball rolling down the surface from a randomly chosen point on the landscape. Although it possible that the ball would keep rolling through small 'bumps' in the error surface, it will most likely rest somewhere along flat regions in the surface, or in one of local minima. Evolutionary search, on the other hand, is in principle much more resistant to the problem of local optima, because (i) its search trajectory is not directly bound to gradient and (ii) it maintains a population of individuals that performs a *parallel search*.

To illustrate how a typical EA navigates the search space, let us consider an example, in which a two-dimensional objective function happens to be this time maximized. Figure 2.2 illustrates its fitness landscape and the distribution of individuals in the population at the beginning, in the middle, and at the end of evolutionary search. Provided proper initialization, the population is *diversified*, with the individuals randomly scattered over the search space (Fig. 2.2a). After a few generations of search, selective pressure pushes the individuals towards regions with higher fitness, while variation operators provide means to *explore* the previously unvisited parts of the space. In this particular run, most of the individuals concentrate around two hills that correspond to local maxima, allowing the search to exploit the vicinity of good, though not the best, solutions (Fig. 2.2b). As a result of exploration, however, some individuals happen to climb the highest peak, making a significant progress towards the optimum. These individuals are particularly likely to be selected as parents for the next generation, and in combination with variation operators they facilitate breeding even fitter individuals. This influences the final outcome of evolution, when the search converges to the highest hill (global optimum), with some individuals populating its very top (Fig. 2.2c).

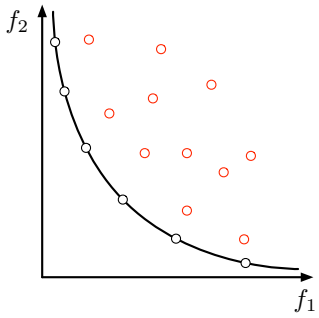


Figure 2.3: The Pareto-optimal front represents the trade-off between competing objectives, in this case f_1 and f_2, both of which are to be minimized. The points on the front are associated with non-dominated solutions and represent the best (lowest) values the can be achieved simultaneously for f_1 and f_2. Red points which lie above the front correspond to solutions that are dominated by other members of S.

Let us note, however, that there are certain caveats involving visualizing the distribution of fitness as a kind of landscape. First of all, the structure of a landscape is contingent on variation operators employed by a search algorithm. For the landscape to be relevant to a search algorithm, these operators must be closely related to those used to induce the neighborhood structure that defines the fitness landscape [275]. Also multimodality of a landscape is not a sole product of an evaluation function; it results from the combination of variation operators and an evaluation function that produces multiple basins of attraction present in the landscape. It may be also worth noting that to determine the elevation of a point in the landscape, one needs to compute the value of objective function, which may be computationally expensive.

Figure 2.2 illustrated that there are two important issues to consider during evolutionary search: population diversity and selective pressure. These factors strongly influence one another as increasing the selective pressure decreases the diversity of a population, and vice versa. In other words, strong selective pressure works towards premature convergence of the search, while a weak selective pressure often makes search ineffective. Maintaining a balance between these two factors corresponds to a trade-off between *exploration* and *exploitation*. Guiding the search towards yet unknown parts of the search space increases diversity and is associated with a high exploration rate, while maintaining the search in the vicinity of known good solutions increases its exploitation, but may prevent search from finding even better solutions. Balancing these factors is essential for an effective evolutionary search. As a matter of fact, both exploration and exploitation are fundamental concepts of any search algorithm [60]. It is often a good practice to start the evolution with a high exploration rate and reduce it over time in favor of an increase in exploitation. EAs are considered good at keeping the balance between those two aspects of search [243].

2.3 Pareto optimality and multiple objectives

In many problems of practical relevance, it becomes necessary to evaluate solutions using multiple objectives. This will also become a necessity in the further parts of this thesis. Whenever there are multiple objective functions, it becomes meaningful to talk about *Pareto optimality*. In such multi-objective problems, there is typically no solution that is better than all others on all objectives, so compromises must be made between the various objective functions [100, 76].

Suppose that the objective functions form the vector function

$$\mathbf{f} = (f_1, f_2, \ldots, f_n) \tag{2.3.1}$$

where $f_i : \mathcal{S} \to \mathbb{R}$. Assume further that each function is to be minimized. A solution $x \in S$ is now said to *dominate* a solution $y \in S$ if it is not worse with respect to any component objective than y and is better with respect to at least one of them:

$$x \succ y \iff \forall i : f_i(x) \leq f_i(y) \wedge \exists j : f_j(x) < f_j(y). \tag{2.3.2}$$

A solution is said to be *Pareto-optimal* in S if it is not dominated by any other solution in S. Pareto-optimal set is the set of such non-dominated solutions, defined formally as

$$S^* = \{x \in S | \nexists y \in S : y \succ x\}. \tag{2.3.3}$$

Pareto optimality comes in handy when one aims at finding a set of candidate solutions that possibly closely approximates the non-dominated solutions and exploits the trade-off between the objectives. A typical Pareto-optimal front is shown in Fig. 2.3. This concept will become indispensable in later chapters, where we will argue for the benefits of multi-objective evaluation.

2.4 Evaluation issues in EC

The effectiveness of evolutionary search is also contingent on the choice of which points in the search space to sample next. In EAs, this choice is typically determined by the combination of individuals in the population, their evaluation function values, the representation used, and the search operators employed. Of these elements, the process of evaluating individuals deserves special attention. An objective function, typically given as a part of a problem formulation, often embodies the requirements set before an ideal solution. It may be then directly used as means to evaluate the 'absolute' quality of a candidate solution, or measure its conformance with the desired behavior as specified by the problem. However, in certain scenarios an objective function may prove too expensive to evaluate, or even impossible to compute in a finite time. Imagine for instance optimizing vehicle parts to meet crash test requirements. In such a case, computing the objective function may involve crashing an actual car, or performing very expensive simulations. In the context of learning game-playing strategies, an objective function could be defined as the expected result over games with *all* possible strategies from the problem's search space. Unfortunately, even for simple games the number of different strategies is computationally intractable (cf. Chapter 5). Consequently, it is a common practice to employ an evaluation function that is intended to guide the search, as we did in this chapter. However, the objective function that often comes as a part of problem formulation may not necessarily be the best tool for this purpose. It is important to realize that it is not always obvious what the 'right' evaluation function for a given problem is. For instance, in the class of *test-based problems* the goal is usually to find a solution that solves all tests (cf. Chapter 5). There will be usually many evaluation functions that are consistent with this objective. The issue of choosing an evaluation function is important for this dissertation and we will analyze it in greater detail in Chapter 6.

Chapter 3

Coevolutionary Algorithms

In this chapter, we focus on the natural phenomenon of coevolution and its artificial counterpart in artificial intelligence. Over the past years, coevolutionary algorithms (CoEAs) have proved to be effective problem solvers for many difficult machine learning problems. We begin the chapter by presenting origins and inspirations of coevolution in Section 3.1, while competitive and cooperative coevolutionary models are introduced in Section 3.2. In Sections 3.3-3.5, we discuss several features specific to CoEAs. In Section 3.6, we demonstrate some of the most impressive applications of CoEAs, and close the chapter in Section 3.7 with the discussion of the current challenges faced by practitioners when attempting to harness the potential of CoEAs for problem-solving purposes.

3.1 Origins

Similarly to traditional evolutionary algorithms, coevolutionary algorithms are inspired by a biological evolutionary process and focus on exploiting the phenomenon of an *arms race* observed in natural environments. In nature, the fitness of an individual within a species results not only from its competition with the other representatives of the same species, but also with other competing species, and with the environment as a whole. Dynamically changing environment requires species to adapt and struggle for both survival and reproduction. When combined, these principles formulate the *natural selection* law described by Darwin [65]. Since natural selection rewards the fittest individuals, evolution of each species moves towards the development of features aimed at outperforming the competition, cooperating with some other species to achieve the mutual goal, or even both[1]. In this way, species exert selective pressure on each other, thus influencing mutual evolutionary development. Ultimately, this allows certain species to evolve traits which give them an edge over other coevolving organisms.

Let us briefly illustrate the principle of competitive coevolution with populations of rabbits and foxes. Faster and smarter rabbits are more likely to escape foxes and survive than slower and dumber individuals, gradually making up more and more of the population. Obviously, the latter individuals may also survive just because they are lucky or because they never encountered any serious threats. Once the surviving population of rabbits starts breeding, we may expect some slow rabbits mate with fast rabbits, fast with fast, and so on. On the top of that, we may witness completely new traits evolve (such as better stealth skills, camouflage etc.), as once in a while nature may mutate some of the rabbit genetic material. On average, the new population of rabbits will be faster and smarter than their parents in the original population because more faster and smarter rabbits survived the foxes. The population of foxes, on the other hand, undergoes a similar

[1] In Nature it is rather hard to tell where competition ends and cooperation starts.

process to adapt to the new environment. Though they may temporarily fall behind, in order to survive they develop over time new traits and skills to close the gap that separates them from their prey. Otherwise, the rabbits could become too fast and smart for the foxes to catch any of them. This principle forms the essence of competitive coevolution and heavily influenced development of CoEAs.

3.2 Coevolution in Computing

In computing science, CoEAs have been introduced as an extension to evolutionary algorithms, where evaluation of individuals is influenced by other evolving individuals [292, 72]. What most coevolutionary methods have in common is the concept of an *interaction* between a pair of individuals. The outcomes of those interactions reveal some information regarding coevolving individuals and are essentially the only feedback that CoEAs receives from the external world. There is no external fitness function in the common sense of this term; rather than that the results of several interaction outcomes together determine an individual's fitness and drive a selection process. The nature of those interactions might be either *cooperative* or *competitive*. Historically, this distinction has led to the development of two major variants of coevolution: *cooperative coevolution* and *competitive coevolution*. In the former case, individuals work together to achieve a common goal. As stated by Potter [293], the cooperative evaluation of each individual in a population is done by concatenating the current individual with the best individuals from the other populations. Cooperative coevolution is one of the few variants of EC that offers some means to decompose a complex problem P into sub-problems $\{p_1, p_2, \ldots, p_n\}$. Thus, such algorithms have been extensively studied in the context of coevolving teams of agents performing a single task, and are well-suited for *compositional problems* [293, 359], where the quality of a solution depends on an interaction among its sub-components. The cooperative coevolution framework has been successfully applied to machine learning benchmarks such as keepaway soccer [356], as well as to real world problems like for instance pedestrian detection systems [42], or large-scale function optimization [367]. The reader interested in more details regarding cooperative coevolutionary algorithms is referred to the works of Potter and De Jong [293], Husband and Mill [138], Krawiec and Bhanu [184].

In case of the competitive form of coevolution, interactions usually take the form of an encounter between two or more competing individuals, and consequently, a gain for one means a loss for the others. Such scenario often imposes a kind of asymmetry on the task setting, giving rise to a different *roles* played by the individuals participating in an interaction. For this reason these individuals are often refereed to as candidate solutions and tests. Depending on the context, a candidate solution might be a program, a machine learning hypothesis or, in case of game-playing agents, a strategy. Accordingly, a test may take the form of an environmental challenge, a training example, or in case of games, an opponent strategy. A single interaction between a candidate solution and a test produces a scalar outcome that reflects the capability of the former to *pass* the latter. In the simplest case, an outcome can be binary or three-state (when ties are accepted); in continuous variant, it reflects the 'degree of passing' the test. The essence of competitive coevolution is that the interactions outcomes drive a search process in the space of candidate solutions and the space of tests, leading to the continuous *arms race* taking place between competing individuals [258]. Thus, a competitive coevolution is particularly useful when an evaluation function is unknown or difficult to define, perfectly fitting interactive domains and test-based problems (cf. Section 5.5.1). In this thesis, we focus exclusively on the competitive form of coevolution.

Algorithm 1 General scheme of a single-population coevolutionary algorithm.

```
1: S ← Initialize Population
2: T ← Select Tests(S)
3: Evaluate Population(S, T)
4: while ¬Termination Condition() do
5:     S ← Select Parents(S)
6:     S ← Recombine And Mutate(P)
7:     T ← Select Tests(S)
8:     Evaluate Population(S)
9: return Fittest Individual(S)
```

3.3 One- and multi-population coevolution

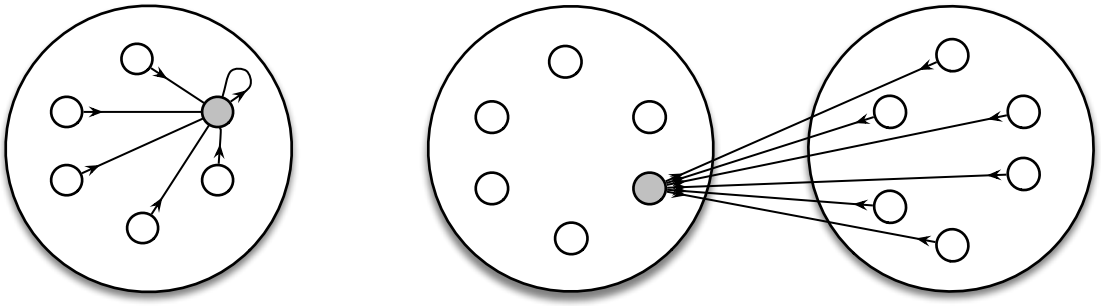
A scheme of a simplest coevolutionary algorithm employing a single population of individuals is shown in Algorithm 1. In such a *single-population* coevolution, individuals serve two roles: they are used as candidate solutions to a problem, and during fitness evaluation their purpose is to provide information regarding other coevolving individuals. Applicability of such a setup is, however, limited to symmetrical domains in which the roles taken by individuals during interactions are interchangeable. Canonical examples of such domains are classic two-player board games like chess, backgammon or Othello (cf. Section 5.4).

In situations where individuals cannot take the role of a candidate solution and a test, it is possible to resort to a *two-population* coevolution. The idea is to maintain simultaneously two populations of individuals — a population of candidate solutions (*learners*) and a population of tests (*teachers*). In contrast to a single-population variant, here the interactions take place only between individuals that belong to different populations. For instance, in the context of game-playing each population provides the opponents used by the other population to evaluate its members. Two-population coevolution is especially effective as a self-adaptive mechanism to increase a problem difficulty as members of the population become more adapt at solving a given problem [292].

Multi-population approach is particularly popular in the cooperative variant of coevolution. Recall from the previous section that the idea behind cooperative coevolution is to decompose a complex problem into several subproblems. Hence, each of the populations in multi-population cooperative coevolution is searching for an optimal subsolution, i.e. a subpart of the complete solution. Though the vast majority of experiments with multi-population coevolution concerns collaborative approaches, there are some attempts of employing multi-population coevolution in the competitive framework, see e.g. [344].

3.4 Differences between coevolutionary and evolutionary approach

Since CoEAs are derived from EAs [96], they share the core mechanics and underlying features, including selection and variation operators (cf. Section 2.1). The fundamental difference between coevolutionary and evolutionary approaches lies in the evaluation phase and concerns fitness assessment. EAs designed to solve optimization problems have direct access to a static *objective* function (2.2.1) that assigns a real value to each individual in $\mathcal{S}$. As a result, given any two individuals $s_i, s_j \in \mathcal{S}$ it is easy to compare $f(s_i)$ with $f(s_j)$ and objectively assess which is more fit (better). By contrast, CoEAs do not use any direct metric of an individual's quality, but instead rely on a *subjective* evaluation function [353] that reflects a relative performance of individuals with respect to other coevolving individuals. In the most common scenario, coevolving



(a) (b)

Figure 3.1: A round robin interaction scheme in single-population (a), and two-population coevolution (b). An individual marked in gray interacts with every member of the same population, or the members of the other population.

individuals engage in interactions, and outcomes of those interactions are typically aggregated to a single scalar value that drives a selection process. As a result, fitness assigned to an individual is *context-sensitive* and subject to the state of constantly changing population. In particular, the fitness ranking of individuals can change depending on presence or absence of other individuals in the population, which is not possible in an ordinary EA. Also, an increase in the subjective fitness does not necessarily imply an increase in the objective quality of individuals. In general, the relation between the subjective and objective fitness of an individual can be extremely complex. Guaranteeing objective progress is thus one of the main challenges in CoEAs (cf. Section 3.7).

Although the fitness landscape for an individual changes dynamically as a function of the population, CoEAs are designed to adaptively focus on the relevant areas of a search space. This property is particularly useful when a problem space is very large or even infinite. Most importantly though, it makes CoEAs particularly well-suited to so-called *interactive domains* in which there is no intrinsic objective function given, or such a function is costly to compute. For more details regarding interactive domains, we refer the reader to the work of Popovici *et al.* [292]. In Chapter 5, we explore test-based problems, which are closely related to interactive domains. CoEAs naturally fit this class of problems as further discussed in Section 5.5.1.

3.5 Interaction patterns

Regardless of whether interactions occur on the level of a single or multiple populations, CoEAs employ an *interaction scheme* that determines which individuals are confronted with each other during fitness evaluation. Of several methods designed for this purpose, probably the most frequently used in practice is a *round-robin tournament* (a.k.a complete mixing [232]). In this scheme each member of population simply interacts with every other individual as illustrated in Fig. 3.1. In the single-population coevolution (Fig. 3.1a) all other members of the population participate in interactions resulting in $m(m-1)/2$ interactions per generation, where m is the population size. In the two-population variant (Fig. 3.1b), on the other hand, all members of the opposite population are employed as interaction partners. In such a case, the number of interactions to be made in each generation depends entirely on the number of individuals in the opposite population. Not surprisingly, the round-robin tournament has the disadvantage of being computationally expensive. To reduce the overall evaluation cost, one may resort to other interaction schemes such as k-random opponents method [297], where an individual interacts with k opponents drawn at random from the current population (nk interactions). Angeline and Pollack proposed the single-elimination tournament [7], which requires $m-1$ interactions, but only the tournament winner has its fitness

computed precisely. Fitnessless coevolution [147] uses the outcomes of interactions to directly drive the tournament selection. This method involves $(k-1)m$ interactions, where k is the tournament size. A detailed review of other methods for selecting interactions can be found in [7, 268, 327]. Let us also note that in Section 12.6, we propose a novel, matrix factorization-based interaction scheme that significantly accelerates the round-robin tournament selection, while maintaining roughly the same level of evaluation accuracy.

Individuals in CoEAs typically engage in multiple interactions giving rise to multiple outcomes of those interactions. In such a case, its important to determine how these outcomes should be *aggregated* to give individual's fitness. The simplest and probably the most commonly used in practice approach is to compute a sum or average of the individual outcomes, and then assign it as fitness. Other, slightly more complex approach known as *competitive fitness sharing* takes into account outcomes of other individuals to weight the interactions and provide more robust evaluation [303, 304]. Arguably, the biggest advantage of such scalar approaches is that they are convenient to use and compatible with the traditional parent-selection techniques. Let us however point out that aggregation of interaction outcomes incurs inevitable information loss and any performance measure based on aggregation is prone to compensation — two individuals may obtain the same fitness even though they solve entirely different subsets of tests. We elaborate more on this and related issues in Chapter 6.

3.6 Applications of coevolution

Historically, one of the first works introducing the idea of coevolution into the field of AI were Samuel's experiments with self-learning methods on the problem of learning game strategy for the game of checkers in 1959 [310]. However, it was not until early 1990s that CoEAs became immensely popular, contributing to rapid developments in this area. The seminal work of Hillis [132] demonstrated a highly effective approach to evolution of sorting networks by using an opposing population of unsorted data sets (number arrays) to test a network's sorting capability. In this case, individuals in both populations engage in a predator-prey type relationship, where candidate solutions representing potential sorting networks are assigned a fitness score based on how well they sort the number arrays in the other population, and the individuals in the second population are rewarded each time an opponent sorting network is unable to sort the related data set. The goal of the algorithm is to produce the smallest network possible that correctly sorts any given data set. Other influential works from this period include Axelrod's research on evolution of cooperation in Iterated Prisoner's Dilemma [12], and Sim's work on coevolution of virtual creatures that compete in physically simulated three-dimensional worlds [327].

Games have always been a particularly popular test-bed for artificial and computational intelligence and, not surprisingly, some of the most popular applications of competitive coevolution involve learning game-playing strategies. In CoEAs, individuals typically serve two roles: they are used as candidate solutions to the problem, while simultaneously they provide evaluation information about other coevolving individuals. The problem of finding a good game-playing strategy fits this scenario perfectly since strategies can be tested by playing against other strategies. One of the most successful application of CoEAs in such a setting is Blondie24, a master-level checker player coevolved by Chellapilla and Fogel without using *a priori* domain knowledge [46, 47, 93]. Evaluation function in Blondie24 employs alpha-beta search algorithm, and a feed-forward neural network for a board evaluation. Individuals in CoEA encode weights for a 5-layer networks, and are evaluated by playing games against each other. Fairly straightforward CoEA employed in this experiment produced a world-class checker player that is competitive with the best human and AI

players. Following this successful application of coevolution to checkers, a similar approach was used to coevolve a chess player, Blondie25, that is competitive with some of the strongest chess programs [94, 95]. Coevolutionary algorithms have also been applied with great success to Othello [149, 49, 237, 341]. For example, Moriarty and Miikkulainen designed coevolutionary approach to discover appropriate playing strategies against particular types of opponents [253]. More recently, Jaśkowski and Szubert employed Covariance Matrix Adaptation Strategy in competitive coevolutionary setup and found out that the proposed approach scales better and finds superior game-playing strategies when compared to plain (co)evolution strategies [146]. Some research also shows the potential of applying coevolution to small-board version of Go [225, 307, 193], albeit the obtained strategies failed to achieve a master level of play. There are also some notable applications of CoEAs to non-deterministic games such as Texas holdem poker [256] or backgammon [286].

Apart from games, competitive coevolution has been successfully applied to a variety of machine learning problems and has played a vital role in evolving complex agent behaviors. For instance, Paredis improved coevolutionary learning of neural network classifiers with life-time fitness evaluation [271], and subsequently used a similar approach to solve constraint satisfaction problems [270, 272]. Angeline and Pollack demonstrated that competitive fitness functions dependent on the constituents of the population may provide a more robust training environment and foster evolution of better solutions than a regular fitness function [7]. There are also numerous successful application of CoEAs to complex compositional problems, where evaluation involves aggregation or composition of certain components from the optimized system. Famous example of such an application is Husbands and Millis' work on job-shop scheduling [138], where individuals encode plans for managing jobs involving the processing of a particular item produced by the shop, and separate populations are used to optimize these plans for each item. Other interesting applications of CoEAs include density classification task in cellular automata [157, 260, 273], design of Boolean networks [329], and protein sequence classification [261].

3.7 Challenges in coevolutionary algorithms

Despite many successes in solving complex problems, CoEAs are known to suffer from so-called *coevolutionary pathologies* — a well-established issue in their design. Coevolutionary pathologies are undesired phenomena, unique to coevolutionary systems, that hinder progress of the search towards a desired goal state or high-quality solutions. Common variants of those pathological behaviors include:

- *Disengagement* [43, 44] occurs when a coevolutionary system decouples, i.e., one population begins to easily outperform another to the extent that its no longer possible to discriminate individuals according to their relative fitness. As a result, any feedback between the coevolving populations is eliminated, leading to loss of selection gradient and causing populations to drift [353].

- *Forgetting* [90, 89] entails the process of losing certain traits (measurable aspects of an individual) acquired during coevolutionary search and subsequently discarded, only to discover that at some point later they are needed in order to obtain a gain in fitness. Some traits may also disappear from a population for instance due to a drift or narrow selective pressure. The issue of forgetting becomes particularly severe when a solution to a problem involves retaining potentially informative, though inferior in quality individuals.

- *Overspecialization* [353] also known as *focusing* is closely related to the more general problem of overfitting in machine learning and occurs when individuals become too focused on solving

only some parts of the problem while ignoring the others. It is particularly likely to manifest itself when individuals start to excessively exploit weaknesses of their opponents, thereby evolving in an unexpected (and often unwanted) way. Overspecialization often results in brittle solutions that fail to generalize [61, 37].

- *Cycling* [55, 324, 37] stems from the relative nature of fitness assessment and can be observed when recurring changes trap a coevolutionary system in some part of a search space. Lack of evolutionary 'momentum' or inherent intransitivity of a problem domain may also significantly contribute to forcing an algorithm into a repetitive cycle.
- *Mediocre stable states* [5, 285] are related to *collusion* [37] and can be observed when competing populations arrive at a sub-optimal equilibrium rather than being engaged in a progressive arms race. This problem is analogous to the problem of convergence to a local optima in regular EAs. It may also happen than coevolving populations improve their fitness, however they do not improve in absolute terms; this phenomenon is often referred to as the *Red Queen effect* [273].

Coevolutionary pathologies constitute a major challenge in the design of effective CoEAs. Over the last decade, much of the research effort has been devoted to understanding their nature and origin [157, 287]. Crucially, it has been shown that coevolutionary pathologies are deeply associated with the use of a relative fitness measure during evaluation and selection, and the aggregation of interaction outcomes commonly performed by CoEAs to guide the search process has been identified as one of their primary sources [288, 86]. In Chapter 6, we discuss issues related to such scalar evaluation functions and identify the evaluation bottleneck whose consequences significantly contribute to the presence of the above pathologies.

Chapter 4

Genetic Programming

The idea of evolving computer programs dates back as far as the early 1950s when pioneer Alan Turing envisioned machine intelligence in his early works. In 1958, Friedberg attempted to solve simple problems by teaching computer to write short programs that were able to manipulate 64-bit data vectors [104]. In the 1960s and 1970s much of the work by Lawrence J. Fogel and John Holland, who are considered one of the earliest practitioners of the genetic programming (GP) methodology, was inspired by the idea of evolving executable structures [134]. Although their work is sometimes considered as inception of program evolution, it is John R. Koza who is widely recognized as the father of modern, tree-based genetic programming. In the 1990s, he pioneered the application of GP in various optimization and search problems and demonstrated how such problems could be solved by genetically breeding populations of computer programs [171, 174, 178, 172].

In the following, we introduce a canonical variant of GP, as originally designed by Koza. Next, we thoroughly discuss the key underlying components of a GP algorithm including representation (Section 4.2) and initialization of the population (Section 4.3), the evaluation and selection of candidate programs (Section 4.4), and the genetic operators of crossover and mutation (Section 4.5). After these introductory sections, we discuss contemporary challenges in GP research (Section 4.7) and demonstrate some of the most impressive human-competitive results obtained by GP (Section 4.6).

4.1 Introduction

GP is a variant of evolutionary computation designed to automatically solve an entire *class* of problems without requiring a user to provide the form or structure of the solution in advance. This is the key conceptual difference between GP and other metaheuristics that tend to operate on a fixed representation of candidate solutions. By contrast, GP searches the space of data structures commonly interpreted as computer programs (*programs* for short). The programs may encode arbitrary procedures as well as complete algorithms for various tasks including classification, regression, feature engineering etc. The concept of a program delivers therefore an unmatched level of flexibility in expressing solutions to virtually any problem, regardless of whether it involves learning, optimization, or some other task. GP offers also an effective way to search the space of programs by relying on an evolutionary algorithm that borrows its mechanics from biological evolution. To a great extent, GP's success is therefore attributed to the combination of its expressive representation and reliance on a powerful paradigm of evolutionary computation.

In general terms, GP is often characterized as a stochastic generate-and-test approach to inductive *program synthesis* [176]. Program synthesis refers to automating the process of generating

computer programs, and describes automatic programming systems that offer the possibility of generating programs directly from a *specification* given in various forms. A specification may take various forms, including formal clauses; in GP however, it is typically assumed to be pairs of input and associated output. GP approaches program synthesis as an optimization problem and employs evolutionary search to iteratively improve candidate programs.

GP belongs to a large family of evolutionary algorithms, and as such its overall workflow is quite similar to that of a regular EA (Fig. 2.1). There are, however, subtle differences on which we elaborate in the following. GP maintains a population P of candidate programs, typically represented as trees (cf. section 4.2). P is initialized with randomly generated candidate programs at the very beginning of an evolutionary run. In each iteration, the quality (fitness) of each program in the population is determined by an evaluation function. Unless evaluation reveals an ideal program, the search continues and GP produces a new generation of programs using the bio-inspired operations of selection, crossover, and mutation. The cycle repeats until the ideal program is found, or any other termination condition is met.

GP has several properties that make it a unique problem solving technique. First of all, candidate solutions are generated from a predefined set of elementary *instructions* that perform some form of computation when interpreted. These instructions are usually problem-specific, and may be equally well associated with certain actions, such as moving an arm of a robot, rather than operating on some variables or registers. What truly matters though, is that the outcomes of these instructions are usually propagated forward (i.e., outcomes of the computation conducted in some program fragment are used by other fragments of the same program), so that subsequent instructions may treat them as input. Instructions therefore have a direct impact on each other. What follows from this observation is that GP is also quite unique when it comes to evaluation of candidate programs. Programs in GP have to be *executed* in order to compute their output. Execution involves processing a series of instructions and, besides a final output returned by a program, it leads to a series of intermediate states of program execution that result from processing certain parts of the program. This observation gave birth to, among others, a new paradigm of behavioral program synthesis [183]. These and other details of representation and evaluation are further discussed in sections 4.2 and 4.4.

Over the years, GP has been extended in many ways, for instance by multi-objective evaluation and selection, maintaining internal archives of well-performing programs, or employing various advanced search operators. More recently, semantic genetic programming has become a particularly active area of research [242, 19, 348, 190, 277, 252]. Semantics of a program p is a vector of the outputs produced by p for the examples in the training set. Such a characterization of program's behavior opens the door to defining semantic-aware population initialization, selection and search operators. The reader interested in these and other extensions of GP is referred to the introductory textbooks on GP [18, 284, 181] and the online bibliography of GP papers [199].

Let us briefly note at this point that GP is not the only paradigm of program synthesis. The other two main paradigms are *deductive program synthesis* [235] and *inductive programming* [322]. Deductive program synthesis relies on a theorem-proving approach that combines techniques of unification, mathematical induction, and transformation rules. The key assumption is that specification of a programming task is complete and given as a relation between the input and output of the desired program. Much of deductive synthesis' appeal comes from deriving programs that are *correct by construction* [170]. Unfortunately, its usefulness in practice is severely limited by the power of automated proof systems.

In contrast to deductive program synthesis, in inductive programming a task specification is not required to be complete and may be given as a list of tests consisting of exemplary inputs to a

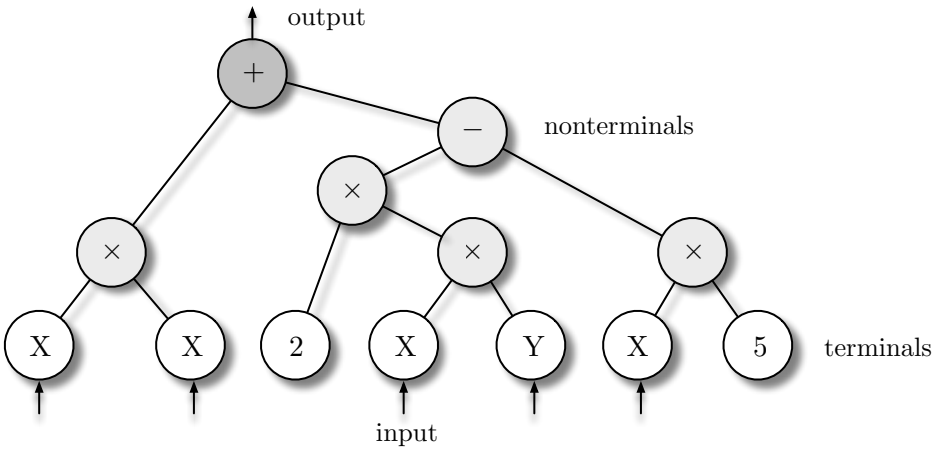


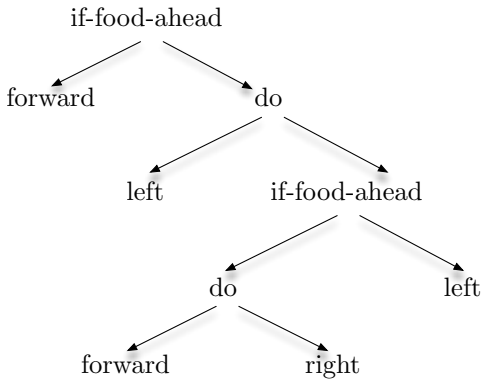
Figure 4.1: Program tree representation in GP. Arbitrary programs are represented by abstract syntax trees, here the tree corresponds to the program $x^2+2xy-5x$. The terminal and nonterminal nodes are marked in white and gray, respectively.

program and corresponding desired outputs. A method performing inductive synthesis is expected to derive a program that passes not only the given list of tests, but also *generalizes* to unseen inputs, i.e., returns a correct output for any input not covered by the specification. Inductive logic programming [322, 323] is the main representative of inductive programming. Inductive logic programming is strictly logic-oriented and operates by induce first-order Horn clauses from a set examples. In this sense, it closely resembles the paradigm of learning from examples, and shares its underpinnings with supervised machine learning [248]. Inductive logic programming approaches program synthesis using generalized rule-learning and is concerned mostly with logic-based programming languages such as Prolog. In the following, we limit our attention to GP as our main paradigm of program synthesis. A reader interested in more details regarding deductive program synthesis and inductive programming is referred to the works of Shapiro [322, 323], Plotkin [283] and Manna [235].

4.2 Representation

Evolving executable structures such as computer programs makes the choice of *representation* particularly important for the ultimate success of a system. In principle, evolving programs could be represented as lines of code in a conventional programming language, however textual representations are cumbersome to handle and would inevitably result in GP producing many syntactically incorrect candidate programs due to stochastic nature of variation operators. For this reason, Cramer [59] proposed to use a parse tree representation which guarantees that only syntactically correct programs are created. The use of parse also allows one to abstract from many language-specific aspects of language syntax, originally introduced to increase human readability. Last but not least, the parse tree representation is more suitable for manipulation by genetic search operators. More importantly, by ensuring syntactic correctness of generated programs, the search space of candidate programs is significantly smaller.

In the parse tree representation originally proposed by Cramer, a subtree never returns a value to the calling statement, but instead designates a command to be executed. Koza [176], on the other hand, proposed to represent programs as *abstract syntax trees* (AST), where the expression computed by a subtree is returned to the calling node in a tree. This seemingly minor extension reinforced a syntax tree as an independent computational unit by providing a direct mechanism for



(a)

```
if foodAhead then
  forward();
else
  left();
  if foodAhead then
    forward();
    right();
  else
    left();
```

(b)

Figure 4.2: Exemplary program tree for Artificial Ant problem (a), and it's implementation in pseudo-code (b).

communication between the nodes in a tree. The popularity of GP as a machine learning system that evolves tree structures is largely due to the immense influence of Koza. It thus not surprising that most of the work in the field is with various tree-based systems, and ASTs became the most common representation in GP.

In GP trees, function calls appear as nodes in a tree, and the call arguments are given by its descendant nodes (children). Consider for instance the tree in Fig. 4.1, containing the mathematical expression $x^2 + 2xy - 5x$. This is the syntax tree of a simple program which calculates this expression. Variables and constants are leaves in the tree and are often referred to as *terminals*. Variables provide a program arguments (inputs), while constants remain the same in every execution.

Crucially, programs in GP do not have to represent algebraic or logical expressions: they can actually *do* things rather than simply perform computation and return values. An example is the tree depicted in Fig. 4.2a, which represents a program that controls an agent, an artificial 'ant' that moves on a discrete two-dimensional grid covered with food. The operator *if-food-ahead* takes two other operators as inputs and executes one of them if there is food ahead, and the other if there is not. The *left* and *right* operators turn the ant ninety degrees in the respective direction, and the *forward* operator moves it one step ahead, simultaneously consuming any food the ant comes across. There is also the *do* operator that executes its children nodes sequentially, one after another. The objective is to evolve an ant controller that collects as many food pieces as possible. This is the artificial ant problem, also known as Santa Fe Ant Trail, originally developed by Jefferson [57], and later popularized by Koza [176]. In Fig. 4.2b, we also illustrate a pseudo-code implementation of the tree in Fig. 4.2a.

Functions and terminals together form the *primitive set* which is used to build candidate programs in GP. The elements of a primitive set are also sometimes referred to as *instructions*. An important property of a primitive set is that each instruction should be able to handle all values it might receive as input. More formally, a primitive set is said to have the *closure property* if all instructions are type-consistent and safe to evaluate [176, 284]. This consistency is crucial for a GP system because evolutionary search operators often exchange arbitrary nodes in a tree. It is thus essential that any randomly created subtree can substitute any node in a tree. Type consistency is always satisfied when a domain and codomain of instructions are of the same type. This is the case, for instance, when evolving arithmetic or Boolean expressions.

Type requirements can sometimes be weakened by providing an automatic conversion mechanism between them. Alternatively, type consistency might be forced by type-aware operators that do by construction cannot violate the constraints imposed by an instruction set. This approach is implemented in, e.g., strongly-typed GP that directly enforces data type constraints [250].

Evaluation safety is the second component of the closure property, intended to address the problem of functions failing at run time. The most popular example of an instruction that is not safe to evaluate is the division operator that fails in case of division by zero. To circumvent this, it is common to employ protected variants of instructions. For instance, protected division behaves just like a regular division except for zero-denominator inputs. In that case, the instruction returns a constant, typically assumed to be 1 [176, 284, 336].

For a GP system to work effectively, a primitive set of consideration should also be *sufficient* for a given problem. A primitive set Φ can be described as sufficient only if it is possible to express a solution to the problem using the functions and terminals in Φ. This requirement is sometimes difficult to meet in practice, because we typically lack necessary knowledge regarding the structure and/or components that comprise an ideal program. An example of a sufficient primitive set in the Boolean domain is $\Phi = \{\texttt{AND}, \texttt{OR}, \texttt{NOT}, x_1, x_2, \ldots, x_N\}$, since any Boolean function of the variables $x_1, x_2, \ldots, x_N$ can be expressed using the elements of Φ. An example of a primitive set that is insufficient for synthesis of transcendental functions is $\Phi = \{+, -, \times, /, 0, 1\}$. For instance, $sin(x)$ is transcendental and cannot be expressed as a rational function comprising any finite compositions of the instructions in Φ. Nevertheless, GP operating on an insufficient primitive set is still capable of developing programs that approximate the desired output, and in many cases the found solution is good enough to meet user's demands.

Past GP research has demonstrated that programs may be represented in ways other than trees. Several alternative program representations have been introduced, including linear GP in which programs are sequences of instructions [31], PushGP where programs are represented as nested lists of instructions that operate on stacks [334], and Cartesian GP which encodes programs as graphs of instructions with edges that determine the data flow [246]. Vast body of research has already established the efficiency of systems that employ these alternative representations. However, in this thesis we limit our attention to the most popular tree-based GP. It is nevertheless worth emphasizing that the algorithms proposed in Chapters 9 and 10 are not restricted by the choice of representation.

4.3 Population initialization

Similarly to other evolutionary algorithms, in GP the initial population is created in a randomized way. Individuals, represented as expression trees (Section 4.2), are generated from a given set of instructions which appear as nodes in a tree. Over the years, many different initialization techniques have been proposed; in the following, we focus on the two most popular ones, known as *Grow* and *Full* [176]. Before we introduce these methods in greater detail, let us first remind that by the *height of a tree* we understand the depth of its deepest leaf, i.e., the length of the longest path starting from the tree's root node to a leaf. The *depth of a node* is defined as the number of edges from the node to the tree's root node, and *the size of a tree* is interpreted as the total number of nodes in this tree.

Equipped with the above definitions, we may now proceed to characterization of Grow and Full which share the same signature and expect the maximum tree height h, the set of terminals Φ_t, and the set of all functions Φ_f to be passed as arguments. In the Full method, nodes are selected randomly only from Φ_f until a node is at the maximum depth. Once the maximum depth

Algorithm 2 Full initialization method for a program tree.

Require: $h \in \mathbb{N}^+$, $\Phi_f \subset \Phi$, $\Phi_t \subset \Phi$

1: **function** FULL(h, Φ_f, Φ_t)
2: **if** $h = 1$ **then**
3: **return** PICKRANDOM(Φ_t)
4: $t \leftarrow$ PICKRANDOM(Φ_f)
5: **for** $i = 1 \ldots$ ARITY(t) **do**
6: $t_i \leftarrow$ FULL($h - 1, \Phi_f, \Phi_t$)
7: **return** t

Algorithm 3 Grow initialization method for a program tree.

Require: $h \in \mathbb{N}^+$, $\Phi_f \subset \Phi$, $\Phi_t \subset \Phi$

1: **function** GROW(h, Φ_f, Φ_t)
2: **if** $h = 1$ **then**
3: **return** PICKRANDOM($\Phi_f \cup \Phi_t$)
4: $t \leftarrow$ PICKRANDOM(Φ_f)
5: **for** $i = 1 \ldots$ ARITY(t) **do**
6: $t_i \leftarrow$ GROW($h - 1, \Phi_f, \Phi_t$)
7: **return** t

is reached, only the terminals from Φ_t can be chosen. In consequence, the generated trees have all leaves at the same depth, and every branch reaches the maximum depth (see Algorithm 2).

Even though the causes all leaves to be at the same depth, it does not necessarily lead to all trees in the initial population having same size. As the instructions in Φ_f may vary in the number of arguments they accept, the trees may differ in their final size and/or shape. Unfortunately the syntactic diversity of programs generated using the full algorithm is often rather limited.

The Grow algorithm, builds trees in the depth-first manner by selecting nodes in each step from Φ_f *and* Φ_t. Once a branch contains a terminal node, that branch is considered complete, even if the maximum depth has not been reached yet (see Algorithm 3). Because the choice of whether to pick a node from Φ_f or Φ_t is random, the trees initialized using the Grow algorithm are likely to be irregular in shape.

The trees produced by both algorithms never exceed the maximum tree height. Nevertheless, using them to obtain trees with certain desired properties, e.g., size, can be tricky. For instance, the grow method is highly sensitive to the size of Φ_f and Φ_t. If there are significantly more terminals than functions, the grow method will most likely generate very short (shallow) trees regardless of the depth limit. If, on the other hand, the number of functions is notably greater than the number of terminals, then Grow behaves similarly to Full.

Numerous past research has shown the significance of diversity in the initial population for an effective GP search. In some applications building full trees may be more desirable than in others, but typically neither Full nor Grow have the capacity to provide satisfactory variety of candidate programs, often producing a fairly biased distribution of trees. For this reason, Koza [176] proposed a combination of these two called *ramped half-and-half*, intended to enhance the diversity in the initial population by constructing half the trees using Full and the other half using Grow. The procedure is repeated for a range of depth limits, hence the term 'ramped'. Suppose the population size is set to 80 and the ramp is set from 2 to 5. The procedure runs in such a case for depth limits 2, 3, 4, and 5, and for each limit, half of the trees (10) is initialized with Grow and the other half with the Full method.

Over the past decade, many different initialization techniques have been proposed, but none gained the same level of popularity as those proposed originally by Koza. In [230], Panait and Luke surveyed and compared several other tree generation algorithms. Of these, PTC2 [229] stands out as designed to enable greater level of flexibility by allowing the user to request trees of specific size. In short, PTC2 picks a random unfilled child node location, fills it with a nonterminal, and repeats this until the number of nonterminals plus the number of unfilled node positions is greater than or equal to the requested tree size. Then the remaining node positions are filled with terminals.

There is also a great variety of initialization techniques in semantic GP, where program semantics are explicitly taken into account when conducting an evolutionary search. For instance, Pawlak and Krawiec [276] proposed semantic geometric initialization, and showed that it leads to superior performance of GP search, i.e., better best-of-run fitness and higher probability of finding the optimal program. More details regarding these and other methods in semantic GP can be found in [275].

4.4 Evaluation and selection

The objective of a GP system is to synthesize a program that meets a specification provided as a set of tests T. Each test is a pair $(in, out) \in T$, where $in \in I$ is the input fed into a program, and $out \in O$ is the corresponding *desired output*. When posed in this way, a program synthesis can be considered as a machine learning task defined within the paradigm of learning from examples. The set of tests T plays then the role of a training set where each test corresponds to a training instance. As in most supervised learning tasks, the set of tests T is typically only a sample from a potentially infinite universe of tests and cannot be assumed to enumerate all possible program inputs and outputs.

A solution to a program synthesis task is a program p that behaves exactly as required by the tests, i.e., for every test $(in, out) \in T$, it returns the desired output: $p(in) = out$. To conduct a search for the ideal program, GP rephrases program synthesis as an optimization problem in which the objective is to minimize (or maximize) a given fitness function f that expresses the degree to which a candidate program attains the search goal. More specifically, fitness function in GP measures how well programs in the population predict the outputs from the inputs, or, in other words, how well they conform with the desired behavior specified by examples in the training set. Assuming, without a loss of generality, that f is minimized, solving a program synthesis task with GP boils down to finding a program:

$$p^* = \arg\min_{p \in P} f(p). \tag{4.4.1}$$

The purpose of a fitness function is thus to give a feedback to the learning algorithm regarding the quality of candidate programs that can be directly translated into the odds of becoming a parent and contributing offspring to the next generation. On the top of that, a fitness function is also expected to help navigate the search space by indicating which regions of the space contain the better (more fit) programs. Co-domain of a fitness function is typically defined on a scale that is ordinal or real-valued, i.e. $f : P \to \mathbb{R}$. In the following chapters, we will often refer to f as an *evaluation function*.

In order to compute fitness, programs in GP have to be *executed*. In tree-based GP, programs such as the one in Fig. 4.1 execute by first reading the input variables, and then repeatedly evaluating intermediate nodes of the tree. Each node reads the outputs of its children nodes and computes its own output. Once all nodes of the tree have been evaluated, the execution terminates and the root node returns the program's output. The discrepancy between the desired value and the value computed at tree's root, aggregated over the set of training examples by means of, e.g., sum of absolute errors, determines program's fitness.

For conformance with its 'mother field' of evolutionary computation, it is rather common in GP to characterize programs with scalar fitness. In contrast to, e.g. black-box optimization, where little more than candidate solution's fitness is available, fitness in GP stems from program's interaction with multiple tests. Crucially, the outcomes of those interactions are easily available,

at no extra cost, to a search algorithm. This observation will become relevant later when defining interaction matrices (Section 8.2).

Formally, the fitness $f(p)$ of a program p assessed on a set of m tests T of the form (in, out) is usually defined as the number of failed tests:

$$f_d(p) = \sum_{(in,out)\in T} [p(in) \neq out], \tag{4.4.2}$$

where $p(in)$ is the output produce by p for in, and $[\cdot]$ is the Iverson bracket. In the above definition, f_d counts the number of failed tests by p but it could equally well count the number of *passed* tests, in which case it would be maximized. In such a case $f_d(p)$ is said to measure a *success rate* or the number of hits. The exact conformance with the behavior specified by the set of tests T is not always critical for the success of a GP system. In continuous domains, where the canonical example is symbolic regression (see also 5.4.5), f will usually compute some form of error, e.g. the mean absolute deviation:

$$f_c(p) = \frac{1}{m} \sum_{(in,out)\in T} |p(in) - out|. \tag{4.4.3}$$

When compared to f_d (4.4.2), f_c *relaxes* the notion of passing a test and allows programs to be 'approximately correct' with respect to a particular test.

The fitness function may be summarized as defining the criterion for ranking candidate programs and for selecting them for inclusion in the next population. The programs in the current population are evaluated relative to a given measure of fitness, with the most fit programs selected probabilistically as seeds for producing the next generation. Selection procedures are typically probabilistic so that inferior programs in terms of fitness still stand a chance of being selected. As a result, the fittest program in the population is not always guaranteed to be selected, and similarly, the worst will not necessarily be excluded. Better programs will typically contribute more offspring that inferior programs. In *fitness proportionate selection* [15], also known as roulette wheel selection, the probability that a candidate program will be selected is proportional to its own fitness and inversely proportional to the fitness of the other programs in the current population. Other methods for using fitness to select individuals have been proposed as well. For instance, *tournament selection* randomly samples k individuals with replacement from the current population into a tournament and selects the one with the best fitness as a parent. Selection pressure in tournament selection can be easily tuned by tweaking the tournament size; the larger the tournament size, the higher the selection pressure.

The selected programs are used as the basis for creating new programs by applying genetic operations of mutation and crossover which we define in detail in the next section.

4.5 Mutation and crossover

GP generates successor programs by repeatedly mutating and recombining parts of the best currently known programs. At each step of the algorithm, a population is updated by replacing a fraction of the population by the offspring of the most fit current programs. In this sense GP closely follows into the footsteps of traditional EAs. However, it quickly departs from the well-established path when it comes to their implementation. In the most popular variant of crossover in GP, known as *tree-swapping crossover*, the offspring is created by replacing the subtree rooted at the crossover point in a parent tree p_1 with the subtree rooted at the crossover point in a parent tree p_2, where the crossover points are randomly chosen nodes. Figure 4.3 illustrates a typical crossover operation. It is also possible to create a variant of crossover that instead returns two offspring, but this is not commonly used in practice.

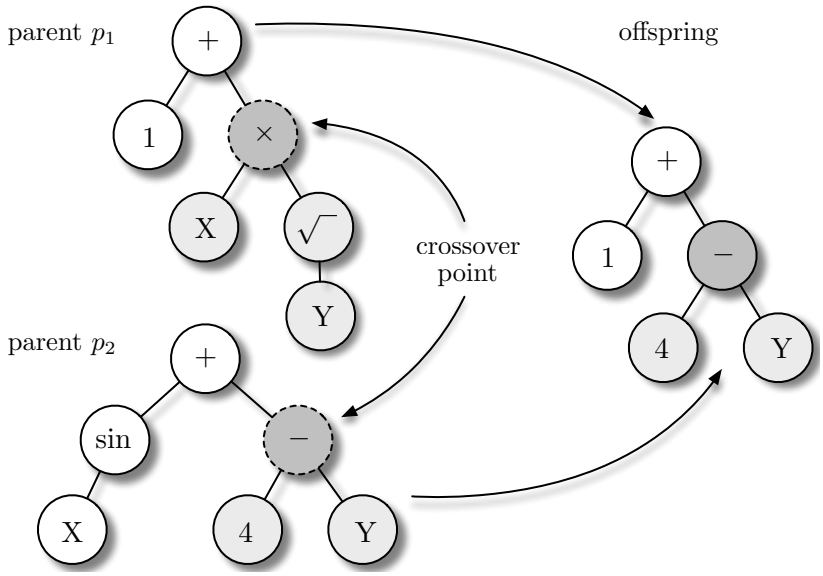


Figure 4.3: Tree-swapping crossover applied to two parent program trees (left). Crossover points (nodes shown in dark gray) are chosen at random. The subtrees rooted at these crossover points are then exchanged to create a child tree (right).

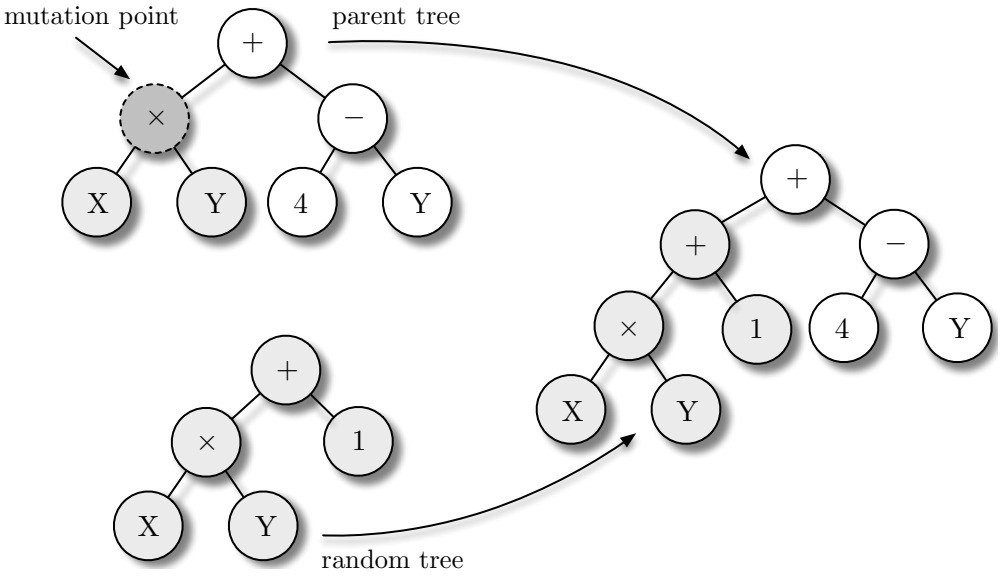


Figure 4.4: Subtree mutation applied to a parent program tree. Mutation point (dark gray node) is chosen at random. The subtree rooted at the mutation point is replaced by a randomly generated subtree.

The probability of selecting crossover points is often *not* uniform. To illustrate why, imagine a perfect binary tree of height h. The probability of choosing a leaf node as the crossover point in such a tree is $\frac{2^h}{2^{h+1}-1}$, which quickly converges to 0.5 with h. Similarly, in most trees generated from a typical GP primitive set, the number of leaf nodes will be roughly equivalent to the number of nonleaf nodes. If the crossover points were chosen uniformly, nearly half of the points would be leaves, limiting so the effect of crossover to simply swapping two leaves. Consequently the uniform selection of crossover points leads to crossover operations frequently exchanging only very small subtrees. It is therefore a widely accepted practice to choose nonleaf nodes 90% of the time and leaves 10% of the time [176].

Mutation in GP operates on a single individual, and its most commonly used variant is known as *subtree mutation*. Subtree mutation randomly chooses a mutation point in a tree and replaces the existing subtree at that point with a new randomly generated subtree, as illustrated in Fig. 4.4. The new subtree is created in the same way, and subject to the same limitations (on depth and/or size) as the trees in the initial population. One may for instance resort to Grow, or Ramped Half-And-Half procedures to generate a subtree. Subtree mutation introduces novelty into a population (in the sense of introducing a completely new piece of code that may be absent there) and is sometimes used to counter premature convergence and the lack of diversity in later generations of an evolutionary run. Due to its similarity to tree-swapping crossover, subtree mutation is sometimes implemented as a crossover between the parent tree and a randomly generated tree. This method is also known as *headless chicken crossover* [6].

Another form of mutation in GP is *point mutation*, which can be likened to the bit-flip mutation commonly used in genetic algorithms [115]. Point mutation simply replaces the primitive implemented by a node with a randomly selected primitive of the same arity. If there are no suitable primitives in the primitive set, the mutation has no effect. In contrast to subtree mutation which modifies exactly one subtree, point mutation is typically applied on a per-node basis, i.e., each node in a tree is altered as previously described with a certain (low) probability. This way, multiple nodes can be mutated within the same application of point mutation.

In contrast to traditional EAs, where offspring are typically obtained via a composition of operators, GP uses the available operators independently, one at a time. One may of course apply mutation directly after crossover, however this is rarely used in practice because tree-swapping crossover highly resembles mutation in its effect. . The choice of which operator to use is controlled by operator rates. For instance, the crossover rate in GP is typically around 90%, while the mutation rate is considerably smaller, and usually does not exceed 10%.

Occasionally, the *reproduction operator* is also used alongside with crossover and mutation. Reproduction inserts a copy of the selected parent tree directly into the next population. It is arguably the simplest possible breeding operator and its sole purpose is to secure the survival of the already good programs. Notice also that even when reproduction is not explicitly used, unaltered programs may be copied to a new population whenever an operator produces a program that does not satisfy the required constraints e.g. on the maximal allowed tree depth. Also, a modification of a program may be ineffective (syntactically or, even more often, semantically).

4.6 Applications of genetic programming

By combining principles of biological evolution with the expressive power of computer programs, GP forms a powerful paradigm of computational intelligence. Already shortly after its inception, Koza summarized the use of GP in several complex tasks such as designing electronic filter circuits and classifying segments of protein molecules [180, 177, 179]. Since then GP algorithms have been successfully applied to a variety of learning tasks and to many optimization problems. Among these are hundreds of tasks of great scientific and practical nature. The most spectacular of them, where GP attained human-competitive[1] performance, are featured at the annual *Humies* competition which is held at the Genetic and Evolutionary Computation Conference (GECCO) since 2004. An excellent example of awarded research is work by Lohn, Hornby and Linden [221], who used GP to automatically design an antenna for NASA's Space Technology 5 spacecraft. The evolved

[1] An automatically obtained solution is considered human-competitive if it is at least as good as a solution designed by human. The detailed criterion for human-competitiveness is available at http://www.human-competitive.org

antenna not only met demanding specification requirements, most notably the combination of wide bandwidth and wide beamwidth for a circularly-polarized wave , but also outperformed the antenna designed by human engineers. The former scored 93% efficiency (compared to 38% of the human-designed antenna) in a series of tests conducted in an anechoic chamber.

Another notable application of GP is the work by Spector [332], who presented examples of GP-based synthesis of quantum computer programs. The specific problems considered in this research concern the communication capacity of certain quantum gates. In these problems, the task is to transfer information from one set of qubits to another, without any direct connection between the two sets of qubits, apart from a single instance of the gate under investigation. Spector also demonstrated how genetic programming with PushGP can be applied to evolve quantum algorithms dedicated to the group of problems which involve verification of certain properties of Boolean quantum gates. Examples of such problems include Grover's database search problem and the Deutsch-Jozsa problem. Fitness of candidate programs was assessed with quantum computer simulations performed via *QGAME* — Quantum Gate And Measurement Emulator.

In [335] Spector *et al.* also described how GP can be applied to a problem in pure mathematics, in the study of finite algebras. The results demonstrate human-competitive results in the discovery of particular algebraic terms, including discriminator, Pixley, majority and Mal'cev terms. GP was shown to exceed the performance of every prior method in finding these terms, both in terms of time and size, by several orders of magnitude.

Unsurprisingly, GP is also used to aid software engineers in writing code. For example, GP has been shown to generate an optimized garbage collector for the C programming language [300] and to evolve new hashing algorithms [20, 84, 161]. Orlov and Sipper [263] introduced a system for evolving Java byte code, where the initial population is seeded [200] with human-written programs already compiled into byte code. The great advantage of the system is that the newly generated programs can be run immediately, without compilation, thanks to Java virtual machine and just in time (JIT) compilers. GP has also been successfully used to evolve small programs from scratch that accomplish real tasks. For instance, White, Arcuri and Clark [355] demonstrated how to evolve pseudo random number generators.

GP has had some great successes when applied to enhance existing software written by humans. One of the best known applications in this area is automatic bug fixing [8]. In the Humies-winning work by Forrest and Weimer [101, 354], GP is combined with program analysis methods to repair bugs in off-the-shelf C programs. The proposed approach does not require formal specifications, program annotations or any special coding practices. Instead, GP is employed to search for program variant that avoids the faulty behavior and retains required functionality. This approach was later extended to be effective on programs with several millions of lines of code [205], while Schulte showed that defects can even be fixed at the level of the assembler code [318]. These results sparked interest in automatic bug-fixing, leading to many contributions in this area [30, 2], including even a hybrid approach of GP with a coevolutionary algorithm [360]. Langdon *et al.* developed a successful GP framework for manipulating source code written in C++ and applied it to, among others, MiniSAT, a popular Boolean satisfiability (SAT) problem solver [281, 282], and computer vision algorithms written for the CUDA architecture [201, 202]. In both cases, they reported a significant speedup in terms of runtime with respect to the original code.

Games have always served as a valuable test-bed for AI methods. Koza was among the first who used GP to evolve strategies for simple two-player games [175]. Since then, GP has been proved effective for learning game playing strategies by successfully tackling various more complex games and game-oriented challenges. One formidable example of the latter is BrilliANT [148], the winner of Ant Wars competition organized as a part of GECCO'2007. BrilliANT was evolved using a

combination of GP with competitive coevolution. Another notable example is evolution of FreeCell solvers using policy-based GP [81]. FreeCell is a highly challenging game for humans, widely recognized as one of the most difficult domains for classical planning. The developed algorithm turned out to scale well and perform better than any other method designed by humans, and the evolved strategy outperformed all humans at a major FreeCell website[2]. Using similar techniques, Hauptman *et al.* [124] achieved human-competitive level of play in the Rush Hour puzzle. Some efforts have been made to apply GP to video games. In [106], J.J. Merelo and his co-workers proposed a framework to evolve complete strategies for StarCraft which are competitive with human-designed bots. The same authors also introduced evolutionary approach to deck building game Hearthstone that involves defining a personalized deck of cards before the actual game [107]. Other interesting examples of GP applied to games include Sipper's work on backgammon [14] and chess [123], Luke's work on evolving soccer softball team [233, 228].

There are also numerous successful applications of GP in biology and medicine. For instance, GP has been shown to be a powerful tool in genome analysis. In [158], Kamath *et al.* describe an evolutionary-based system that proved effective at identifying meaningful features for splice sites in DNA sequences. These features are too complex for humans to design manually, but they are necessary to obtain high accuracy and precision in splice site identification and annotation. Another interesting research was conducted by Widera *et al.* [357] who evolved energy functions for the protein structure prediction. The best evolved function was obtained by combining energy terms designed by human experts, and outperformed the energy functions optimized by the Nelder-Mead algorithm. In medical imaging, Krawiec and Pawlak applied GP to the problem of detection of blood vessels in ophthalmology [189]. The obtained results suggest that a properly configured GP algorithm can be a viable alternative in similar medical and non-medical detection tasks.

The above examples clearly illustrate that GP reaches well beyond automatic programming and program synthesis. The vastness of GP literature is also illustrated by the number of entries in the genetic programming bibliography, which has already surpassed 11,000 mark [199].

4.7 Challenges in genetic programming

As illustrated in this chapter, GP builds upon genetic algorithms in order to enable evolution of computer programs. It has been demonstrated to produce human-competitive results in many applications such as simulated robot control or recognizing objects in visual scenes. Nevertheless, despite numerous successes, synthesis of fully-fledged programs by means of GP remains a challenging task. The most obvious reason is the huge size of the program space that grows exponentially with the length of considered programs (or the number of tree nodes in the tree-based GP; see Section 6.1 for a more concrete example of this issue). Furthermore, the relationship between program code (syntax) and its effects (semantics) is extremely complex: a minute modification of a program can drastically change its behavior, i.e., the output it produces. It is also likely, however, that a large modification leaves program's behavior intact, because the same behavior can be implemented by many programs — as a matter of fact, there are usually infinitely many programs that behave in exactly the same way. In consequence, so-called fitness landscape [165], i.e., the fitness function plotted against the search space, is very rugged in GP, features many local optima as well as plateaus (see also Section 2.2).

Another non-trivial challenge in GP stems from the fact that the conventional fitness functions (Eqs. 4.4.2 and 4.4.3) are characterized by low fitness-distance correlation [345]. In this context,

[2] http://www.freecell.net

distance is typically understood as the number of search steps required to reach the optimal solution to a problem at hand. Evaluation function in GP does not correlate well with the measures of syntactic similarity between programs. This applies to generic distance measures [262], as well as to operator-based distance measures, like for instance crossover-based distance [117]. For this and other reasons on which we elaborate in Chapter 6, the conventional fitness functions in GP are not necessarily good tools to guide the search process towards the most promising parts of the search space. This issue is prevalent in almost all bio-inspired metaheuristic algorithms that employ aggregative fitness function as yardstick of candidate solution's quality, and is of central interest to this thesis.

The most commonly used search operators in GP include mutation and crossover that perturb a candidate program by randomly modifying its components (Section 4.5). Despite their conceptual appeal, these operators perform a blind *syntactic* search that is neither very efficient in terms of convergence to an optimal solution, nor scales well with a problem size. However, it is the meaning of a program i.e., program *semantics*, that determines its quality, and that meaning, is largely neglected in conventional GP. This issue has been identified and characterized from different perspectives by referring to concepts like locality [275], meant as the degree of distortion introduced by the genotype-phenotype mapping. Put in these terms, genotypes (program code) map into phenotypes (program behavior) at low locality, so that even a small change of code can translate into a huge difference in its behavior.

Programming task specification is usually incomplete, i.e., the set of tests does not contain all correct input-output pairs. Nevertheless, GP is expected to generalize beyond the set of training examples (and in this sense perform induction). Successful generalization typically goes hand in hand with avoidance overfitting. Nevertheless, achieving good generalization is not simple, in particular when the set of tests is small. As a result, GP algorithms find it hard to scale well with task difficulty and instance size. For example, one of many common benchmarks in GP is parity task [238], where each test is a list of bits of fixed length accompanied by the parity bit. A program solving this task for lists of length 5 can be relatively easily synthesized with a baseline, unsophisticated variant of GP. However, synthesizing a program that realizes this functionality for lists of length 7 is already much harder [277], and for 10 or more bits becomes extremely difficult.

Chapter 5

Test-Based Problems

Many problems addressed by evolutionary computation require candidate solutions to be evaluated multiple times to accurately estimate their performance. This may be necessitated by the presence of noise in the evaluation process or randomness in the underlying simulation environment [154]. Apart from that, in some problems, the quality of a candidate solution can be determined only by evaluating it against a number of *tests*. A common example of such *test-based problems* [70, 144] are games, where the set of tests are opponents.

In this chapter, we formalize the class of test-based problems that proves particularly useful when modeling domains with no intrinsic objective measure giving a value to candidate solutions. We first detail the key elements of this definition in Section 5.1, with particular focus on the concept of interaction between a candidate solution and a test. Afterwards, in Section 5.4 we discuss several examples of such test-based problems, and finally in Section 5.5 we concern ourselves with issues specific to approaching these problems with coevolutionary algorithms and genetic programming.

5.1 Definition

The task of optimization entails making decisions which are optimal with respect to some objective function. The objective function measures the quality of candidate solutions and embodies the aspiration to make the best possible decisions. Optimization problems are often divided into two categories depending whether decision variables are discrete or continuous. In combinatorial optimization the set of feasible solutions is discrete, and decision variables can take values from bounded, discrete sets of elements. The situation is quite different in continuous optimization where the goal is to find the optimal setting of continuous decision variables.

The common ground of combinatorial and continuous optimization involves searching for a solution that maximizes (or minimizes) a given objective function. However, there exists a class of optimization problems for which the objective function is so complex that evaluating it becomes computationally intractable. For example, consider searching for the optimal game-playing strategy in the game of Go, an ancient Eastern board game that has puzzled AI researches for decades. Although quite recently, in a major breakthrough for AI, the computer program AlphaGo has beaten a top human player in a five-game match [325], Go is still considered one of the most difficult problems in AI. If one aims at maximizing the odds of winning against any opponent, the objective function in Go could be defined as the expected outcome of games with all possible opponent strategies. In such a scenario, evaluating a given candidate solution would involve playing games with all possible Go strategies, which is computationally infeasible even for much simpler

games than Go. For instance, the number of unique game strategies in the apparently trivial game of Tic-Tac-Toe played on the tiny 3×3 board is the staggering 3.47×10^{162} [145].

As illustrated by the above example, in some of the most challenging problems in AI, obtaining the exact value of objective function is computationally intractable. This may stem from many causes, however in this thesis, we are interested in those illustrated above with the presence of innumerable opponents, i.e pertaining to *test-based problems* [35, 67, 289, 144], where the performance of a candidate solution is determined by the outcomes of multiple *interactions* with *tests*. An interaction between a candidate solution and a test produces a scalar outcome that reflects the capability of the former to *pass* the latter. Typically, the set of tests is large, making it infeasible to evaluate candidate solutions on all of them. In the literature, test-based problems are also referred to with terms like interactive domain [292], adversarial problem [164], or competitive fitness environment [7, 228].

In this thesis, we will use the following formal definition of a test-based problem:

Definition 5.1. A test-based problem is a tuple $(\mathcal{S}, \mathcal{T}, g)$, where:

- $\mathcal{S}$ is a set of candidate solutions (*solutions* or *individuals* for short),
- $\mathcal{T}$ is a set of tests with which the candidate solutions interact,
- $g : \mathcal{S} \times \mathcal{T} \to \mathcal{O}$ is an interaction function, where $\mathcal{O}$ is a totally ordered set, and
- a solution concept that determines the goal of search.

Therefore, in contrast to traditional optimization problems, test-based problems lack an objective evaluation function. As a substitute, the outcomes of interactions between the elements of $\mathcal{S}$ and $\mathcal{T}$ have to be used to either promote or demote certain candidate solutions. The nature of those interactions may be rather simple, such as comparing the moves of two players in the rock-paper-scissors game, or more complex, as in the case of simulating and dispatching rules for scheduling in job shops given several floor plans (in which the floor plan forms a test). Whenever possible, we abstract away from such details and concern ourselves only with the outcomes of interactions.

In the following chapters, we assume that $\mathcal{S}$ and $\mathcal{T}$ are finite, and that the interaction between a candidate solution and a test is deterministic. Let us also note that when $\mathcal{T} = \{t_1\}$, any test-based problem is reduced to an optimization problem with the objective function $f(s, t_1) = g(s, t_1)$. When $\mathcal{S} = \mathcal{T}$, we talk about *symmetrical test-based problems,* where the role of candidate solutions and tests are played by the same entities. Problems that are not symmetrical are typically called *asymmetric test-based problems* [261, 292].

The definition of test-based problems can be conveniently used, for instance, to phrase the problem of finding the best strategy in the game of chess for the player who plays blacks. In such a case, $\mathcal{S}$ includes all possible black player strategies, the $\mathcal{T}$ consists of all white player strategies, and the interaction function g corresponds to playing an actual game of chess between a black and white strategy. To identify that best strategy, one may seek for a solution that maximizes the expected result of interaction with a randomly selected test, so-called expected utility solution concept, which we discuss in greater detail in the following section.

As the above examples suggest, test-based problems are typically associated with applications in games. Indeed, a game-playing agent must be usually tested against many opponents in order to assess its quality, and the number of such opponents is often very large [292]. The class of test-based problems is however much wider. Programs evolved in genetic programming are usually applied to multiple *fitness cases* (see also Section 5.5.2). When evolving controllers that, e.g., maintain the balance of an inverted pendulum or streer a mobile robot, it is common to perform multiple

simulations that vary in initial conditions or other parameters (see [284] for review). Also, whenever the evaluation of candidate solutions is stochastic or involves noise, their robustness needs to be assessed in multiple scenarios that vary in the realization of the underlying random variable(s). For example, a physics simulator used to assess the performance of a robot (candidate solution) in an environment (test) may use randomness to emulate the effects of friction. Clearly, the class of test-based problems embraces a vast range of problems.

5.2 Extensions and related concepts

While Definition 5.1 is sufficiently general for the purpose of this thesis, we find it necessary to discuss possible directions in which it can be further extended.

Test-based problems are in fact a special case of *co-search problems* in which there are just two domain roles: candidate solutions and tests. In general, co-search problems many involve n domain roles and n sets of entities X_i that play these roles. The interaction function is then of the form $g : X_1 \times X_2 \times \cdots \times X_n \to \mathcal{O}$, where $\mathcal{O}$ is a partially ordered set. For a large majority of co-search problems that have been studied in practice, the outcome set $\mathcal{O}$ is a subset of $\mathbb{R}$, and thus *completely* ordered; however, that requirement is not strictly necessary. The common ground of co-search and test-based problems is the notion of solution concept that indicates which locations in the search space, if any, constitute solutions to these problems. We elaborate more on solution concepts in the next section.

Co-search problems in which entities from domain roles are combined together to build candidate solutions are known as *compositional problems*. Problems that belong to this category do not feature any tests to provide information about the solutions. Instead, to build a candidate solution one must use components from each entity set X_i corresponding to each domain role. As an analogy, think of a football team which is not complete unless there is a goalkeeper, a defender, a midfielder, etc.

A co-search problem can be further generalized to a *co-optimization problem*. In place of a solution concept, a co-optimization problem specifies an (in general partial) order on the candidate solutions and defines the search goal as finding a maximal element of the order. Note that a co-optimization problem can be easily converted into a co-search problem by defining the solution concept to be the set of maximal elements according to that order. In that sense, co-optimization generalizes co-search, and is arguably a more refined notion. For more details regarding co-search and co-optimization problems, a reader is referred to the works of Popovici and de Jong [292, 291, 290].

Notice also that particular elements of Definition 5.1 resonate with the concepts in game theory [110]. For instance, interaction function g can be interpreted as a payoff matrix, where $\mathcal{S}$ and $\mathcal{T}$ correspond to sets of game strategies, while the entry corresponding to a row s and a column t is $g(s,t)$. We may also translate the interaction outcomes in test-based problems into *payoffs* granted to candidate solutions and tests, since the former are expected to perform, while the latter should be informative, i.e., provide information regarding capabilities of candidate solutions. Examples of domains of this sort are common in game theory [114, 88], where payoffs are assigned differently depending on which role an entity plays. Games considered in game theory typically assume numerical payoffs, and for instance, in *zero-sum games* we have that $g(s,t) = -g(t,s)$, where $g(s,t)$ is interpreted as the outcome s receives for its interaction with t, while $g(t,s)$ denotes the outcomes assigned to t from its interaction with s.

5.3 Solution concepts

The above generalizations of a test-based problem illustrate the vast range of problems that can be defined over interactive domains. On its own, however, an interactive domain provides merely a way to perform an interaction between two or more entities. Outcomes of these interactions are not enough in the context of problem-solving. Only when coupled with an explicit yardstick that decides which entities in the domain are better than others, an interactive domain becomes either a co-search or co-optimization problem.

In general, solving a test-based problem consists in finding a candidate solution with certain properties captured by a *solution concept* [89], which articulates the goal of search. A solution concept is a formalism that originates in game theory [264] and specifies which elements of a search space are solutions to a problem. A solution may be a single candidate solution or comprise a *subset* of them; in either case, a solution concept defines the properties such a formal object must meet. Much of the theoretical work has analyzed algorithms designed for test-based problems in terms of solution concepts they implement [89, 87, 289], showing the importance of understanding and correctly choosing the right solution concept for the problem at hand.

A solution concept is sometimes defined as a subset of a set of *potential solutions* that are built from a set of candidate solutions. The notion of potential solutions adds another layer of abstraction which is particularly useful in problems (e.g. compositional problems mentioned earlier) where no single candidate solutions is a well-formed, feasible solution. For instance, in multi-objective optimization one could be interested in finding the entire non-dominated front of candidate solutions (or find a good approximation of it). A potential solution becomes then a subset of $\mathcal{S}$, and the set of all potential solutions is a power set of $\mathcal{S}$. Another example is evolution of complex objects such as teams of cooperating agents, where a potential solution is composed of agents who collaborate as a team.

For the purpose of this thesis it is sufficient to assume that individual candidate solutions are potential solutions. We are interested exclusively in domains where a single candidate solution that solves the problem may exist. It is sometimes useful to think of a solution concept as partitioning of the candidate solutions in $\mathcal{S}$ into *actual solutions* and non-solutions. We denote the actual solutions as a subset $S^+ \subset \mathcal{S}$.

Let us also point out that a solution concept does not provide any means to *compare* candidate solutions. Given only a solution concept it is thus not possible to determine if a candidate solution s_1 is preferred over a candidate solution s_2. For this reason, it is common to resort to a *preference relation* $\preceq$ defined on $\mathcal{S}$ such that if $s_1, s_2 \in \mathcal{S}$, $s_1 \preceq s_2$ is interpreted as $s_1 \in \mathcal{S}$ is not worse than $s_2 \in \mathcal{S}$. An obvious choice for a preference relation is to define it as $s_1 \preceq s_2 \iff Q(s_1) \leq Q(s_2)$, where Q is the *quality function* that measures, for instance, the average score received over all possible tests. In this case, the preference relation determines the total ordering of solutions. The most popular solution concepts used in the literature include Best Worst Case, Maximization of Expected Utility, and Pareto optimal set [89, 71, 292]. In the following, we briefly characterize these solution concepts and describe the classes of problems for which they are useful.

Best Worst Case

In this solution concept we search for a solution that performs best on the hardest test, i.e, the one that maximizes the minimum possible outcome over interactions with all tests. For this reason, it is particularly useful in domains where the performance in the worst possible scenario is of the highest importance. More formally, let $q(s) = \min_{t \in \mathcal{T}} g(s,t)$, then:

$$S^+ = \arg\max_{s \in \mathcal{S}} q(s).$$

Notice that the hardest test is determined for each s independently. Given $q(s)$, we can easily provide a preference relation that generalizes such Best Worst Case problems to co-optimization problems:

$$s_1 \preceq s_2 \iff q(s_1) \leq q(s_2),$$

where $s_1, s_2 \in \mathcal{S}$. In the literature, Best Worst Case is also known as *maximin*. A very similar solution concept, Worst Best Case, or *minimax*, could be defined analogously.

Pareto Optimal Set

Pareto Optimal Set solution concept phrases a test-based problem as a multi-objective optimization problem, where every test in $\mathcal{T}$ is viewed as a separate objective to be optimized. The goal then becomes to find the Pareto front among the candidate solutions. Therefore a potential solution is a subset of $\mathcal{S}$, and the set of potential solutions is a power set of $\mathcal{S}$. Let us define a non-dominated front of candidate solutions as (recall that we assume g to be maximized)

$$\mathcal{F} = \{s_1 \in \mathcal{S} \mid \forall_{s_2 \in \mathcal{S}} \left[\forall_{t \in \mathcal{T}} \left[g(s_1, t) \leq g(s_2, t)\right] \implies \forall_{t \in \mathcal{T}} \left[g(s_1, t) = g(s_2, t)\right]\right]\}.$$

According to Pareto Optimal Set solution concept, the solution is

$$S^+ = \{\mathcal{F}\}$$

The preference relation between two Pareto-fronts $\mathcal{F}_1$ and $\mathcal{F}_2$ may be built based on the notion of Pareto dominance:

$$\mathcal{F}_1 \preceq \mathcal{F}_2 \iff \forall_{s_1 \in \mathcal{F}_1} \forall_{s_2 \in \mathcal{F}_2} \left[\forall_{t \in T} \left[g(s_1, t) \leq g(s_2, t) \wedge \exists_{t \in T} : g(s_1, t) < g(s_2, t)\right]\right],$$

where $\mathcal{F}_1, \mathcal{F}_2 \in 2^{\mathcal{S}}$.

A major practical limitation of Pareto Optimal Set solution concept is that it may not sufficiently narrow down the set of possible solutions, as many of them tend to be incomparable when the set of tests $\mathcal{T}$ is large.

Pareto Optimal Equivalence Set

Pareto non-dominated front may contain candidate solutions that are indistinguishable according to their interaction outcomes. In other words, two candidate solutions s_1 and s_2 belong to the same *equivalence class* if they receive the same outcomes on all tests in $\mathcal{T}$. The equivalence relation in question can be defined formally as

$$s_1 \sim s_2 \iff \left[(\forall_{t \in \mathcal{T}} g(s_1, t) \leq g(s_2, t)) \wedge (\forall_{t \in \mathcal{T}} g(s_2, t) \leq g(s_1, t))\right].$$

It is further reasonable to assume that, in order to cover the Pareto front, it is sufficient to have a representative of each equivalence class. This observation gave rise to Pareto Optimal Equivalence Set solution concept in which S^+ contains *at least* one candidate solution from each equivalence class of the Pareto Optimal Set. In another variant of this solutions concept known as Pareto Optimal Minimal Equivalence Set, S^+ contains *exactly* one candidate solution from each equivalence class of the Pareto Optimal Set.

Simultaneous Maximization of All Outcomes

In this solution concept, we seek a candidate solution that maximizes its performance on all tests simultaneously. Let us denote a subset of such candidate solutions as

$$\mathcal{H} = \{s_1 \in \mathcal{S} \mid \forall_{s_2 \in \mathcal{S}} \forall_{t \in \mathcal{T}} \left[g(s_1, t) \leq g(s_2, t) \implies g(s_1, t) = g(s_2, t) \right]\}.$$

Notice strong formal similarity of this solution concept to Pareto optimal set. In fact, it is easy to show that $\mathcal{H} \subset \mathcal{F}$ using the rule of distribution of universal quantifiers over implication:

$$\forall_x \left(\varphi(x) \implies \psi(x)\right) \implies \left(\forall_x \varphi(x) \implies \forall_x \psi(x)\right).$$

The set of actual solutions becomes $\mathcal{S}^+ = \mathcal{H}$.

Unfortunately, this solution concept has a limited application scope, as for many problems there does not exist a single potential solution that simultaneously maximizes the interaction outcome against all possible tests.

Maximization of Expected Utility

Maximization of Expected Utility (MEU) solution concept specifies as solutions those elements of $\mathcal{S}$ that maximize the expected score against a randomly selected opponent. Thus

$$S^+ = \arg\max_{s \in \mathcal{S}} \mathop{\mathbb{E}}_{t} \left[g(s, t) \right],$$

where $\mathbb{E}$ is the expectation operator and t is a randomly drawn test from T. For finite Ts, MEU may be viewed as maximizing the sum of outcome values over all tests. As opposed to other solution concepts, MEU features a natural continuous quality function: candidate solution's *expected utility*, i.e., the average outcome against all tests

$$Q_{\mathcal{T}}(s) = \mathop{\mathbb{E}}_{t \in \mathcal{T}} \left[g(s, t) \right]. \tag{5.3.1}$$

A solution in the sense of MEU is an $s \in \mathcal{S}$ such that maximizes $Q_{\mathcal{T}}(s)$ in $\mathcal{S}$. Expected utility can also be conveniently used to define the preference relation in the form:

$$s_1 \preceq s_2 \iff \mathop{\mathbb{E}}_{t} \left[g(s, t) \right] \leq \mathop{\mathbb{E}}_{t} \left[g(s, t) \right],$$

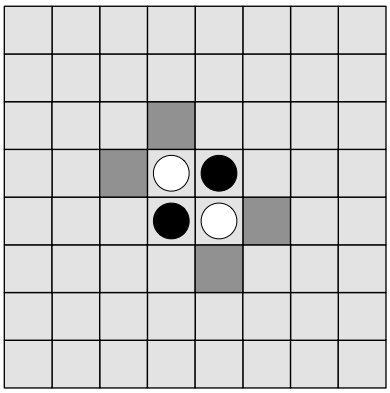
where $s_1, s_2 \in \mathcal{S}$ and $t \in \mathcal{T}$. Notice that the above preference relation induces a total order on solutions in $\mathcal{S}$. In co-optimization, MEU is sometimes referred to as *generalization performance* [52, 50].

Finding a solution that maximizes $Q_{\mathcal{T}}$ is challenging in many test-based problems, because the number of tests in $\mathcal{T}$ is usually large or infinite. This can be mitigated by estimating utility by confronting the solution with a *sample* of tests $T \subset \mathcal{T}$ of a computationally manageable size. This leads to an approximate quality function:

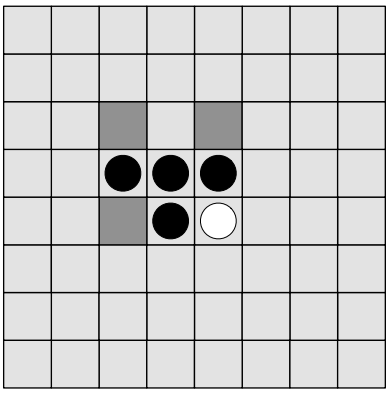
$$\hat{Q}_T(s) = \frac{1}{|T|} \sum_{t \in T} g(s, t), \tag{5.3.2}$$

which is commonly used as a fitness function in evolutionary computation, i.e., $f_T(s) = \hat{Q}_T(s)$. Notice that f_T is an unbiased estimator of $Q_{\mathcal{T}}$ when T is an uniform sample of $\mathcal{T}$. In the following we refer to it as to 'scalar evaluation'.

Even though MEU has been proven to be globally non-monotonic [89] and $Q_{\mathcal{T}}$ tends to be sensitive to the distribution of tests' characteristics (*behaviors*, *phenotypes*) [151], it is very common



(a) Othello initial board state. Black to move.



(b) Board state after black's move. White to move.

Figure 5.1: Othello boards with legal moves marked as shaded locations.

in the literature. In particular, it is often presented as the concept implemented by competitive coevolution, where tests evolve to challenge the candidate solutions and thus force them to improve in quality. It is also popular in real-world studies on, e.g., games (even when not explicitly referred to), because average performance against opponents is often the characteristic sought for in practice (all the above reservations notwithstanding). For these reasons, we treat MEU as our solution concept of choice in all future considerations.

5.4 Examples of test-based problems

Test-based framework introduced in the previous section allows modeling a wide spectrum of practical problems, including those which are traditionally approached as optimization problems. In this section, we introduce several examples of test-based problems. These problems are also a part of our experimental test-bed used throughout this thesis.

5.4.1 Othello

Othello is a perfect-information, zero-sum, two-player strategy game played on an 8×8 board. There are 64 identical pieces which are white on one side and black on the other. The game begins with each player having two pieces placed diagonally in the center of the board (Fig. 5.1a). The black player moves first, placing a piece on one of four shaded locations, which may lead to the board state in Fig. 5.1b. A move is legal if the newly placed piece is adjacent to opponent's piece and causes one or more of the opponent's pieces to become enclosed from both sides on a horizontal, vertical or diagonal line. The enclosed pieces are then flipped. Players alternate placing pieces on the board. The game ends when neither player has a legal move, and the player with more pieces on the board wins. If both players have the same number of pieces, the game ends in a draw.

Despite simple rules, the game of Othello is far from trivial [219]. The number of legal positions is approximately 10^{28} and the game tree has roughly 10^{50} nodes [4], which precludes any exhaustive search method. Othello is also characterized by a high temporal volatility: a relatively high number of pieces can be flipped in a single move, dramatically changing the board situation. These features and the fact that the game has not yet been solved makes Othello an interesting test-bed for computational intelligence methods.

When framed as a test-based problem, $\mathcal{S}$ is the set of all black player strategies, while $\mathcal{T}$ consists of all white player strategies. The interaction function g involves playing an Othello game between $s \in \mathcal{S}$ and $t \in \mathcal{T}$, and its co-domain $\mathcal{O}$ is the ordered set $\{lose < draw < win\}$. It is convenient to assign numerical scores to particular outcomes of a game, i.e.:

$$g(s,t) = \begin{cases} 1 & \text{if } s \text{ wins a game,} \\ 0.5 & \text{if there is a draw,} \\ 0 & \text{if } s \text{ loses a game.} \end{cases}$$

One may also look for the best strategy for the game regardless of the color played; in such a case a single interaction could involve a *double game*, where both agents play one game as black and one game as white player. In each game, one point is to be divided between players: the winner gets 1 point and the loser 0 points, or they get 0.5 point each in the case of draw.

5.4.2 Numbers Games

COMPARE-ON-ALL and COMPARE-ON-ONE are variants of the Numbers Game [353] proposed in [70]. Candidate solutions and tests are points in an l-dimensional space represented as real-number vectors of length l.

In COMPARE-ON-ALL (COA), a candidate solution s and a test t are compared on all dimensions. The interaction function rewards s if it weakly dominates t, i.e., if and only if all numbers in s are greater or equal to the corresponding numbers in t:

$$g(s,t) = \begin{cases} 1 & \text{if } \forall_{i=1,\ldots,l} : s_i \geq t_i \\ 0 & \text{otherwise} \end{cases},$$

where x_i denotes the ith dimension in individual x.

In COMPARE-ON-ONE (COO), s and t are compared only on the dimension that stores the largest number in t, i.e., $j = \arg\max_{i=1,\ldots,l} t_i$. The interaction function rewards s for achieving at least as high value in that dimension:

$$g(s,t) = \begin{cases} 1 & \text{if } s \geq t_j, \\ 0 & \text{otherwise.} \end{cases}$$

Straightforward formulation notwithstanding, both problems are well-known coevolutionary benchmarks [68, 320, 35, 71] and enable objective and precise measurement of search progress. COMPARE-ON-ONE is more challenging in being designed to induce *overspecialization* [353], a coevolutionary pathology in which candidate solutions and tests focus only on some objectives while ignoring the remaining ones. To make progress on this problem, a coevolutionary algorithm has to maintain the tests that support all underlying objectives from the very beginning of the run.

5.4.3 Iterated Prisoner's Dillema

Iterated Prisoner's Dilemma (IPD) is an abstract two-player game involving a series of interactions, each of which is a Prisoner's Dilemma (PD) game. IPD is primarily used to study cooperation in social, economic and biological interactions. It is considered nontrivial in being a non-zero sum game and in its iterative character, making it an attractive playground for competitive environments [53].

In a PD, a player can make one of two choices: cooperate or defect. If both players cooperate, they receive a payoff R, whereas if they both defect, they get a smaller payoff P. Defecting

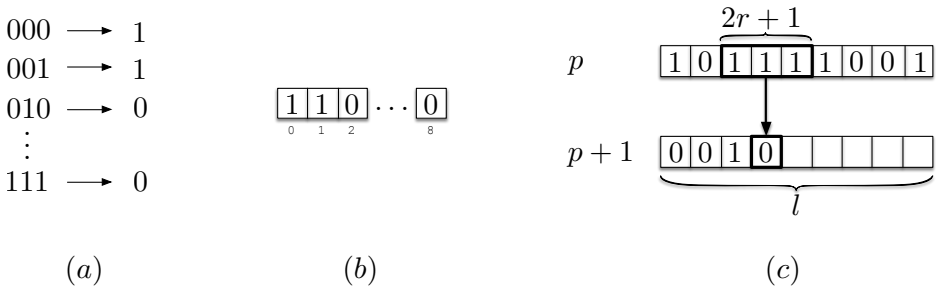


Figure 5.2: An example of state transition in CA: the rule (a) encoded as a bit string (b) is applied to CA window by window at step p in order to obtain CA at step $p+1$ (c).

against a cooperator gives a payoff M which is higher than R, and the cooperator in such a case receives the lowest possible payoff m. In sum, the PD payoff matrix must satisfy two conditions: $M > R > P > m$ and $2R > m + M$ [294].

Following other studies [103, 63, 51, 121], in the experimental part of this thesis we consider IPD extended to multiple choices (moves or *levels of cooperation*) and employ the *memory-one* form of IPD in which players remember their moves from the previous PD iteration only, and we represent strategies as look-up tables [13]. A c-choice IPD strategy is an $c \times c$ matrix M, where m_{ij} specifies player's move to be made given his previous move i and the opponent's previous move j. The other element of player's strategy is the initial move m_{00}, so in total a strategy is represented by $c^2 + 1$ numbers in the range $[0, c-1]$.

A single IPD game between a candidate solution s and a test t involves a series of PD *episodes*. In each PD episode, s makes a move i, t makes a move j, and that brings them the payoffs $2.5 - 0.5i + 2j$ and $2.5 - 0.5j + 2i$, respectively. The outcome of an IPD game is determined by comparing the total payoffs p_s and p_t accumulated over PD episodes:

$$g(s,t) = \begin{cases} 1 & if \sum p_s > \sum p_t, \\ 0.5 & if \sum p_s = \sum p_t, \\ 0 & \text{otherwise.} \end{cases}$$

5.4.4 Density Classification Task

In the Density Classification Task (DCT), the objective is to find a one-dimensional binary cellular automaton (CA) that performs majority voting. A cellular automaton is a discrete model studied among others in computability theory, mathematics and theoretical biology [362, 163, 361]. In DCT, candidate solutions are *rules* that govern the state transitions of CAs, while tests are bit vectors of length l that determine the *initial configurations* (IC) of the automata. The next state of the ith bit is determined by applying the rule to the window that comprises the current bits at positions $i - r$ through $i + r$. A rule is represented as a lookup table. A window of size $2r + 1$ implies 2^{2r+1} possible combinations of bits in a window and the same number of entries in the lookup table. Therefore, the search space comprises $2^{2^{2r+1}}$ rules and there are 2^l possible initial conditions.

The objective is to construct a rule s that causes the CA to converge, within a prescribed number of iterations, to the state of all ones if the percentage (*density*) of ones in an IC t is greater than or equal to 0.5. Otherwise, the rule should cause the CA to converge to the state of all zeros. An interaction between s and t starts with the CA in the initial state determined by t and consists in iteratively applying s to all elements of the current configuration (cf. Fig 5.2c).

5.4.5 Symbolic regression

In symbolic regression [239] the goal is to find a mathematical expression that best fits a given training set. The candidate expressions are created from a predefined set of instructions (building blocks) that typically includes arithmetic operators, elementary functions, constants and input variables (cf. Section 4.2). In contrast to conventional regressions techniques (e.g. linear regression) that search only the space of parameters for a specific model, symbolic regression learns the *form* (formula) of the model from data and optimizes its parameters at the same time. When cast as a test-based problem, the set of candidate solutions $\mathcal{S}$ consists of all expressions formed by combining the available instructions, while the set of tests $\mathcal{T}$ comprises all points from a training set, each of them holding the values of input variables and the corresponding desired output. A natural choice for an interaction function in such a setting is, e.g., the absolute error fitness function (Eq. 4.4.3).

To give a more concrete example, let us consider a symbolic regression problem in which the goal is to create an expression that outputs values returned by the quartic univariate polynomial $f_q(x) = x^4 + x^3 + x^2 + x$ in the range from -1 to 1. Candidate solutions are built from the basic arithmetic functions such as $+, -, \times, /, \exp, \log, \sin, \cos$ as well as one input variable x, while uniformly sampled points x in the range $[-1, 1]$, paired with $f_q(x)$, are used as the set of tests $\mathcal{T}$. The search for an ideal expression (candidate solution) might be performed by means of genetic programming (Chapter 4).

5.5 Algorithms for test-based problems

As we have shown, there is a wide spectrum of problems that can be modeled as a test-based problem. To solve a test-based problem, we seek for an element from solution space $\mathcal{S}$ that conforms a given solution concept. However, we have not yet discussed *how* to find such a solution. As argued in Section 5.1, test-based problems typically lack an objective function that could guide a search algorithm towards good solutions. In absence of an explicit yardstick to evaluate candidate solutions (cf. Section 5.1), an evaluation function can only be expressed in terms of interactions between a candidate and some tests. The outcomes of these interactions are then typically aggregated into a single scalar value that forms its evaluation.

However, in nontrivial test-based problems, the set of tests is typically large or infinite, which precludes evaluating candidate solutions on all of them. Thus, to define a computationally tractable fitness function, one has to limit the number of tests used for fitness evaluation. This issue can be handled by letting tests *coevolve* with candidate solutions as in the coevolutionary framework (Section 5.5.1). Another possibility is to sample a subset of tests T from $\mathcal{T}$ for evaluation purposes. A sample T could be drawn once at the beginning of a run, and then remain fixed, or T could be sampled multiple times, for instance in every generation of an evolutionary run as in [53, 149]. The former case resembles typical GP setup (Section 5.5.2), while the latter has been shown to improve generalization in coevolutionary learning [149].

In the subsequent section, we demonstrate that coevolutionary algorithms are particularly well suited to solve test-based problems. Next, we introduce the test-based perspective on the task of program synthesis (cf. Section 4.1), and illustrate how genetic programming can be employed to solve such problems.

5.5.1 Competitive coevolution

Competitive CoEAs, presented in detail in Chapter 3, are particularly well suited methods for test-based problems because they do not rely an objective performance measure to evaluate candidate solutions, but instead explore outcomes of interactions with other individuals to guide the search. Moreover, CoEAs typically feature two populations[1] that may be identified with the roles of candidate solutions and tests. By this token, they naturally subscribe to the framework of test-based problems.

A typical CoEA maintains a population of candidate solutions $S \subset \mathcal{S}$ and a separate population of tests $T \subset \mathcal{T}$. In every generation, each candidate solution $s \in S$ interacts with every test $t \in T$, producing an interaction outcome $g(s,t)$. The outcomes of these interactions are then used to assign fitness to individuals in S and T. In general, candidate solutions are evaluated for their performance, while tests are rewarded for *informing* about the capabilities of evolving solutions, such as the number of distinctions they make [36]. In this context, solutions act as *learners* while the population of tests plays the role of a *teacher* [23, 83, 157] who, ideally, should pose tests that are neither too hard nor too easy, i.e., feature the level of difficulty that provides a tractable learning gradient [352] for the learners. According to Juillé [156], it is desirable to expose the learners to tests that are *"just a little more difficult than those they already know how to solve"*. Since coevolutionary algorithms attempt to select the tests in an adaptive, dynamic manner, they can in principle avoid potential biases resulting from the use of a fixed sample of tests (whether drawn at random or selected manually).

A critical advantage of CoEAs stems from their generality. In contrast to many specialized methods which are dedicated only to certain sub-classes of test-based problems (e.g., Monte Carlo tree search for games), CoEAs can be in principle applied to any test-based problem, provided they are properly configured. The configuration process entails defining the interaction function and adapting evolutionary search operators to the representation used by candidate solutions and tests.

Interestingly, there is a clear analogy between coevolution and the concept of shaping that originated in research on animal training and behavioral psychology [330]. Shaping typically consists in exposing the learner to a series of training episodes, starting from simpler tasks, and progressively increasing their difficulty. The population of tests in coevolutionary algorithms naturally matches the role of human experimenter, who is responsible for providing training tasks in conventional shaping. An interaction corresponds to a learning episode in shaping, and its outcome characterizes both the learner's capability to solve a test as well as the test's suitability for that particular learner. This feedback can be used by the learner to improve its performance, and by the experimenter to adjust the difficulty of the subsequent tests. This analogy to shaping (similar to *staged/incremental learning* [255]) has been noticed since the early works on coevolution [23], and is periodically revived in the more contemporary studies [79, 339].

Despite the risk of falling victim to so-called *coevolutionary pathologies* (Section 3.7), CoEAs proved effective at solving many nontrivial instances of test-based problems, including learning game strategies [47] and evolving controllers [337]. See also Section 3.6 for a review of applications of CoEAs.

[1] Though for symmetrical test-based problems, one-population coevolution is equally popular.

5.5.2 Test-based genetic programming

In competitive coevolution, the working population of tests changes with time. However, the conceptual framework of test-based problems comes in handy also in the more general setting of an evolutionary algorithm, where the set of tests T is fixed and given as a part of problem formulation, like the training set of examples in machine learning. This is the default setting for genetic programming, where candidate solutions are symbolically represented executable structures like programs or expressions.

Recall from Chapter 4 that the task of automated program synthesis by means of genetic programming can be conveniently phrased as an optimization problem in which the search objective is to find a candidate program that minimizes the objective function f (Eq. 4.4.1).

An ideal program is typically specified by a set of tests T. Each such test is a pair $(x, y) \in T$, where x is the input fed into a program, and y is the desired outcome of applying it to x.

While the given set of tests specifies the behavior of an ideal program, fitness function is the only yardstick of candidate program's conformance with the desired behavior. In essence, fitness functions in GP such as (4.4.2) and (4.4.3) apply a candidate program to a given input and measure how much the actual output diverges from the desired output. The outcomes resulting from repeating this elementary interaction for every program and tests are aggregated into a single value that is intended to reflect program's quality.

In this light, it is easy to notice that a program synthesis task can be formulated as a test-based problem, in which candidate solutions are programs, tests are pairs of a program inputs and the associated desired (expected) output, and passing a test requires a program to produce the desired output for a given input (or an output that is sufficiently close to it). In general, we will assume that an interaction between a program p and a test t produces a scalar outcome $g(p, t)$ that reflects the capability of the former to *pass* the latter.

A GP algorithm solving a test-based problem of program synthesis maintains a population of programs $P \subset \mathcal{P}$. In every generation, each program $p \in P$ interacts with every test $(x, y) \in T$, in which p is applied to x and returns an output denoted as $p(x)$. If $p(x) = y$, p is said to *solve* the test and $g(p(x), y) = 1$. If, on the other hand, $p(x) \neq y$, we set $g(p(x), y) = 0$ and say that p *fails* (x, y). This definition of an interaction outcome is particularly useful in domains where programs return discrete values; in continuous domains, absolute or square error may be more appropriate.

Given this test-based framework, the conventional GP fitness that rewards a program for the number of failed tests (Eq. 4.4.2) can be written as

$$f(p) = |\{t \in T : g(p, t) = 0\}|. \tag{5.5.1}$$

Notice that such formulation corresponds exactly to the solution concept of maximization of expected utility, i.e., the average score on all tests.

5.6 Chapter summary

In test-based problems, candidate solutions interact with multiple tests. Depending on problem domain, tests may take on the form of, e.g., opponent strategies (when evolving a game-playing strategy), simulation environments (when evolving a robot controller), or program inputs and associated desired outputs (when synthesizing programs in genetic programming).

One promise of CoEAs is that they are capable of evolving complex entities given only information on how these entities interact with each other. When applied to test-based problems, CoEAs can autonomously induce a useful search gradient by selecting the tests in an adaptive manner

to pose the right challenge for candidate solutions. As suggested by multiple empirical studies [339, 151, 157, 70], the fact that a set of tests is *dynamically* provided by the second population of individuals that coevolve along the candidate solutions in the first population, allows CoEAs to find good solutions faster and often more reliably, compared to approaches where, if feasible, all tests are used, or otherwise are drawn at random.

The framework of test-based problems allows us to elegantly embrace not only the domains with very large of even infinite numbers of tests, but also domains where evaluating the quality of a candidate solution requires performing interactions with a *static* or entire set of tests. For instance, in program synthesis with GP, the specification of an ideal program typically contains a sufficiently small number of examples to treat them all as tests. The set of tests remains constant during evolution, despite not enumerating all possible input/output pairs to evaluate a candidate program. Nevertheless, as demonstrated in Section 5.5.2, its convenient to express such a problem as a test-based problem, which consist in finding a candidate solution that performs well across the given set of tests. Such a formulation has its own merits — when it is possible to evaluate a candidate solution on all elements of T, the problem may be expressed as a traditional optimization problem, where the exact objective performance of a solution can be calculated as an outcome of its interactions with all tests.

The most important property that makes a problem test-based is therefore not necessarily a large set of tests, or a lack of objective evaluation function, but rather the presence of an interaction function that characterizes the outcomes of interactions between two or more entities.

The above considerations incline us also to informally distinguish test-based problems with small and large number of tests. When the number of tests is small, judging the success of any algorithm attempting to solve a given instance of a test-based problem is a fairly straightforward task because objective performance can be easily obtained. There may be different criteria of success, such as finding the highest quality candidate solution within a given time, or within a certain budget of function evaluations. Depending on the nature of a problem, one may resort to any search algorithm, including evolutionary as well as exact methods like, e.g., branch-and-bound. There is typically no need to use coevolutionary algorithms, even though there are some documented attempts of employing CoEAs in classical function optimization [311].

Conversely, when the number of tests is large, coevolutionary methods typically gain a significant edge over traditional optimization algorithms by offering better performance due to the phenomenon of arms race that takes place between competing individuals [7, 272]. The other key factor that contributes to that is the inherently adaptive nature of competitive coevolution, which constantly monitors the performance of candidate solutions and actively adjusts the set of tests to provide a 'just right' challenge for them. By selecting tests adaptively, coevolution offers also better scalability since only some interactions are indeed performed. Note, however, that employing other optimization methods is still possible by, e.g., sampling a subset of tests $T \subset \mathcal{T}$ [53, 149].

Finally, let us notice that test-based problems often pose difficulties not encountered in traditional optimization, such as maintaining search progress. While we already discussed several such challenges in the context of CoEAs (cf. Section 3.7) and GP (cf. Section 4.7), here we would like to stress that some issues concerning evaluation in test-based problems are pertinent to any evolutionary search algorithm. In particular, regardless of whether a set of tests is static or dynamic, or whether interactions occur between individuals in the same population or in different populations, decisions need to be made as to how the outcomes of those interactions should be translated into individuals fitness. In the following chapter, we argue that the most widely used approach of aggregating outcomes of interactions is not only harmful, but also contributes to several pathological behaviors exhibited in evolutionary search.

Chapter 6

The pitfalls of scalar evaluation

The main motivation for this chapter is the observation that the habit of driving search using a conventional, scalar evaluation function cripples the performance of evolutionary algorithms. In the following, we identify and discuss the consequences of the conventional approach to evaluation of candidate solutions in test-based problems. This chapter is based on the material published previously in [214, 186, 211].

6.1 Evaluation bottleneck

As witnessed in previous chapters, many optimization and learning problems approached in evolutionary computation involve evaluation functions that reward the candidate solutions by counting the number of tests they pass. When evolving computer programs or controllers, passing a test requires producing the desired output for a given input. When learning game strategies, tests are embodied by opponents, and a candidate solution passes a test if it wins a game against it. In these problems, known as test-based problems, candidate solutions need to *interact* with multiple 'environments' in order to be evaluated. In algorithms designed to solve such problems, search is typically driven by an evaluation function that aggregates the outcomes of those interactions. To illustrate this, let us repeat the formula (5.3.2) for the conventional evaluation function that is consistent with the solution concept of maximization of expected utility:

$$f_T(s) = \frac{1}{|T|} \sum_{t \in T} g(s,t). \tag{6.1.1}$$

Clearly, evaluation of a candidate solution s boils down to counting the number of passed tests in T. Similarly, conventional fitness evaluation in genetic programming consists in applying a program to multiple tests and aggregating the observed differences between the actual and the desired program output (Eq. 4.4.2 and 4.4.3).

An evaluation function that counts the number of passed tests usually forms an inherent part of the problem and makes it amenable to many conventional search algorithms that expect a scalar objective. It is arguably not only convenient as a succinct yardstick of candidate solution's quality, but also consistent with the conventional way of posing problems in optimization and machine learning.

On the other hand, scalar evaluation is very crude in its aggregate characterization of the interaction outcomes. Even though some individuals in a population may fare better than others on some tests but not on others (and vice versa), these individual differences are often not reflected in the overall fitness, which informs on the average performance only. Candidate solutions often encode complex entities such as computer programs or game-playing agents, however, scalar

evaluation entails very little information regarding their *characteristics*. All that remains from a nontrivial evaluation process is a single value that indicates the number of passed tests or the total error on a set of tests. This problem is more common than it may appear, as aggregative objective functions prevail in practice.

The main claim of this thesis is that the conventional *scalar evaluation function imposes inevitable information loss, while a richer characterization of candidate solution performance might help making search more effective, not least by providing natural means for differentiation in the population.*

Aggregation of test outcomes causes thus an unnatural *evaluation bottleneck* in the communication between an evaluation function and a search algorithm. There are domains where guiding the search using only scalar evaluation is inevitable. Consider for instance black-box optimization problems where the communication between an objective function being optimized and a search algorithm is limited to exchanging information on a candidate solution's quality [155, 34]. In such problems it is not possible to broaden the evaluation bottleneck by making a search algorithm better informed about properties of candidate solutions. However, this is more exception than a rule. In other domains, where details of evaluation are not hidden, it becomes natural to ask: do we have to 'compress' all the information about interactions outcomes into one scalar value? Why not exploit it more carefully, for the sake of making search more efficient? Wherever possible, a search algorithm should be provided with richer information on solution characteristics and so enable it to perform better. Several past studies followed that intuition (cf. Section 8.6), but little has been done so far to propose a generic, principled approach to address this issue. This observation is the cornerstone of heuristic search objectives proposed in Chapter 8.

Example 6.1. A classical case study often considered in GP consists in synthesizing a function that solves a Boolean multiplexer problem. This task is conceptually based on the behavior of an electronic multiplexer device that decodes a binary address and returns the value of the corresponding data register. The input to a Boolean k-multiplexer function is a bit-string of length n consisting of k address bits and 2^k data bits ($n = k + 2^k$). Solving an n-bit multiplexer problem consists in finding a function $\mathbb{B}^n \to \mathbb{B}$ that returns the correct data bit for all 2^n combinations of inputs. An example of 7-bit multiplexer, together with a valid input vector and the corresponding output, is shown in Fig. 6.1.

The class of multiplexer problems is perceived as an interesting and nontrivial benchmark for machine learning; as stated by Koza [173]:

"Multiplexer functions have long been identified by researchers as functions that often pose difficulties for paradigms for machine learning, artificial intelligence, neural nets, and classifier systems."

The reason behind the complexity of the multiplexer problem is the presence of *epistasis* in input variables, i.e., the fairly complicated interaction pattern between the variables that determines the output value of a multiplexer.

Let us now consider synthesis of an 11-bit Boolean multiplexer. This task can be approached using GP with candidate solutions represented as expression trees (cf. Section 4.2) and instruction set Φ that comprises four binary instructions `{AND, OR, NAND, NOR}`. A minimal potential solution to this problem is a tree with 11 leaves and 10 internal nodes. There are

$$C_{10} \times 4^{10} \times 11^{11} = 16796 \times 1048576 \times 285311670611 \approx 5.025 \times 10^{21}$$

such trees, where C_n is the nth Catalan number defined as [1]:

$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$

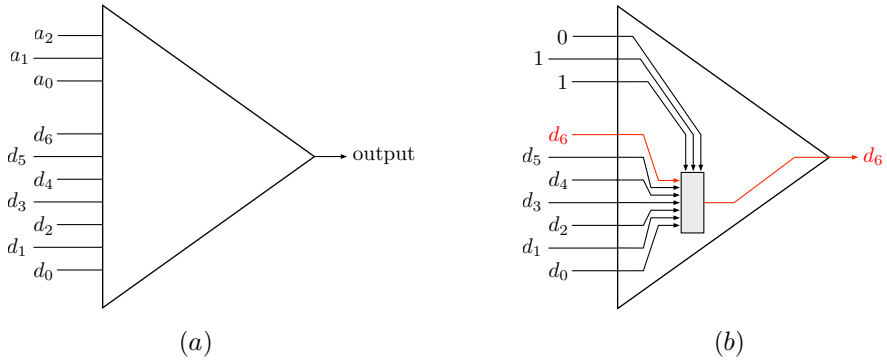


Figure 6.1: The Boolean 7-bit multiplexer; (a) model with three address bits and seven data bits, (b) addressing data bit d_6.

The objective evaluation function for this problem applies the candidate solution to all possible input bit combinations and counts the number of correct output values. Notice that there are $2^{11} = 2048$ possible input combinations so that the objective function takes only 2049 distinct values (0 to 2048 inclusive). The search algorithm therefore navigates the search space of staggering 5.025×10^{21} candidate solutions guided by merely 11 bits of information in each evaluation. Let us recall that for the sake of clarity we consider here the Boolean domain, arguably the simplest one. The problem, however, is more general and by all means not limited to Boolean domain. ■

The above example clearly illustrates the existence of evaluation bottleneck: relying on *any* search algorithm to *efficiently* navigate such a vast search space using a scalar evaluation function which provides low-information feedback is *very* optimistic. Even though 11-bit Boolean multiplexer problem is considered moderately difficult and many GP algorithms converge to a correct solution, there are many similar in complexity problems (e.g. 11-bit Parity) that hardly ever get solved.

6.2 Compensation of interaction outcomes

The most severe implication of the evaluation bottleneck is *compensation* of interaction outcomes: if two solutions pass the same number of tests, they are considered equally valuable, regardless *which* particular tests they pass. Otherwise, one of the compared solutions is deemed better, but, again, disregarding the behavior on particular tests. As a result, solutions can receive the same fitness, even if the results of their interactions with the same tests are completely different. This may render them indiscernible in selection phase, leading to a loss of diversity and a premature convergence.

Compensation could be avoided by comparing the candidate solutions using the *dominance relation*:

$$s_1 \succ s_2 \iff \forall t \in T : g(s_1, t) \geq g(s_2, t) \wedge \exists t \in T : g(s_1, t) > g(s_2, t). \tag{6.2.1}$$

The dominance relation compares the behavior of solutions on particular tests and is in this sense more scrupulous than a scalar objective function. However, it is a partial relation and, in consequence, fails to provide a useful search gradient whenever none of the compared solutions passes a superset of tests solved by the other solution (cf. [182]). This is unfortunately common: for two unrelated solutions, it is much more likely that they are mutually non-dominated than that one of them dominates the other, and that likelihood grows with the number of tests in T.

Scalar evaluation and dominance occupy thus two extremes in scrutinizing interaction outcomes. On of the main motivations behind this thesis is to develop a compromise that inherits the benefits of both approaches. In Chapter 9 and 10, we demonstrate how a useful multi-objective characterization of candidate solutions can be obtained automatically, in a largely data-driven manner.

An evaluation function that counts the passed tests is symmetric with respect to tests and in this sense totally unbiased. Only the count of passed tests matters – which of them a given candidate solution passes is irrelevant. In practice, however, tests in T usually differ with respect to their inherent *difficulty*, which is typically not known *a priori*. In particular, tests may vary in *objective difficulty*, i.e., the probability of being passed by a randomly generated candidate solution, but they often differ also when it comes to *subjective difficulty*, meant as the probability that a given search algorithm produces a solution that passes a test. In Section 7.2, we formalize the concept of test difficulty, show that it can be easily estimated even if $\mathcal{S}$ is large or infinite, and demonstrate that the distribution of objective difficulty among tests is often highly non-uniform. Coming across a problem instance with all tests equally difficult is much less likely than with difficulty varying across the tests. As a result, scalar evaluation functions may become ineffective in guiding a search algorithm towards good solutions because a candidate that solves k easy tests and the one that solves k difficult tests are considered equally valuable during selection. Consequently, an evolutionary search process tends to focus on the easiest tests, often leading to premature convergence and solutions that correspond to local minima in the search space.

The compensation of interaction outcomes affects the internal dynamics of algorithms that use them to drive search, as well as the *post-hoc* comparison of solutions they produce. It is also important to point out that compensation is not exclusive to domains with binary interaction outcomes, where a test can be only passed or failed, but occurs also in the case of continuous outcomes (cf. Chapter 11). Search algorithms that rely on scalar evaluation deliberately ignore these aspects and are thus prone to inferior performance.

6.3 Loss of gradient

The aggregative evaluation function f_T that counts the number of passed tests also falls victim to a *loss of gradient* that occurs when candidate solutions solve the same number of tests. This is particularly likely to happen when the number of tests is small because f_T is inherently *discrete*, and can assume only $n+1$ values for n tests. With such a limited range of values, evaluation leads to coarse-grained fitness that often fails to differentiate solutions, and leaves a selection operator blind to promising candidate solutions. In consequence, the search process becomes severely underinformed about the characteristics of candidate solutions, and pays for it with unsatisfactory performance, limited scalability and, in extreme cases, purely random search.

Unfortunately, seemingly obvious remedy such as increasing the number of tests (whenever possible) is typically insufficient because candidate solutions evolve with time, and once a search process identifies good and thus similarly fit solutions, ties become likely. Also, in the realm of test-based problems, increasing the number of tests only tends to make evaluation more precise and leads to better estimation of solution's generalization performance [53]. However, in presence of a rugged fitness landscape, precision may be have little impact, and evaluation functions based on varying in size subsets of tests may perform equally well (cf. Section 2.2).

The loss of gradient has also severe consequences for CoEAs. When one population of individuals reaches a state where relative fitness diversity dramatically decreases, the other is left with insufficient information to learn from and, in consequence, is unable to progress in a meaningful

way. If the populations remain in such a 'decoupled' state, they typically become polarized in terms of subjective fitness and are driven into arbitrary equilibria [358]. Also, for coevolution to maintain its coadaptive search gradients, a feedback between the populations is essential. In a competitive setting, a classic example of its lack is a game where a champion plays a novice. If the outcome of a game between the two players is the only source of learning experience, it is virtually impossible for a novice player to improve his play based on the experience gathered from such games. The loss of gradient may also affect cooperative coevolutionary algorithms when one or more populations has converged, preventing other populations from making further progress by donating degenerate or faulty components during collaboration.

One may argue that this problem does not apply to continuous domains such as symbolic regression, where the outcome of an interaction between a program and a test is a real value. However, its important to realize that the mapping from the space of programs to the space of their behaviors (genotype-phenotype mapping) in GP is many-to-one, meaning that there will be many syntactically different programs that implement the same target function. In the eyes of a scalar evaluation function, such candidate programs are equally valuable and none of them can be deemed better. For this reason, the problem of discreteness and gradient loss also pertains to continuous interaction outcomes.

6.4 Search bias

The way in which a learning algorithm transforms one candidate solution into another is often viewed as a search though the space of possible candidate solutions. Any stochastic search algorithm (other than a purely random search) has a *search bias* that controls this transformation and increases the probability of visiting some candidate solutions over the others. In terms of EAs, this bias is represented by the mutation and crossover operators. For that instance, the search bias of a genetic algorithm equipped with a single-bit mutation operator inclines it to visit the solutions that are similar (in the sense of Hamming distance) to the solutions in the current population.

As a consequence of diverse test difficulty and search bias, a search algorithm driven by a scalar evaluation measure tends to converge to candidate solutions that solve tests that are easier and better 'reachable'. In parallel search techniques like GP and CoEAs, the probable aftermath of that is premature convergence, which we illustrate in this section. For this purpose, let us first introduce the concept of an *outcome vector* that characterizes the *behavior* of a candidate solution:

$$o(s) = [g(s, t_1), g(s, t_2), \ldots, g(s, t_{|T|})]. \tag{6.4.1}$$

An outcome vector is thus a vector where ones and zeros correspond to passing or failing respective tests. The conventional evaluation function f_T that counts the number of passed tests can be thus reformulated as

$$f_v(s) = \sum_{i=1}^{|T|} o_i(s), \tag{6.4.2}$$

where $o_i(s)$ denotes the ith element of the outcome vector $o(s)$. The above notion of an outcome vector is also closely related to program's semantic in GP (cf. Section 4.1). In accordance with studies on semantic GP [242, 252], by *program semantics* (*semantics* for short) we mean the vector (tuple) of outputs returned by a given program p for a given set of tests $(in_i, out_i) \in T$, i.e.,

$$sem(p) = [p(in_1), p(in_2), \ldots, p(in_{|T|})]. \tag{6.4.3}$$

Semantic GP methods rely on s to, among others, diversify populations and design search operators.

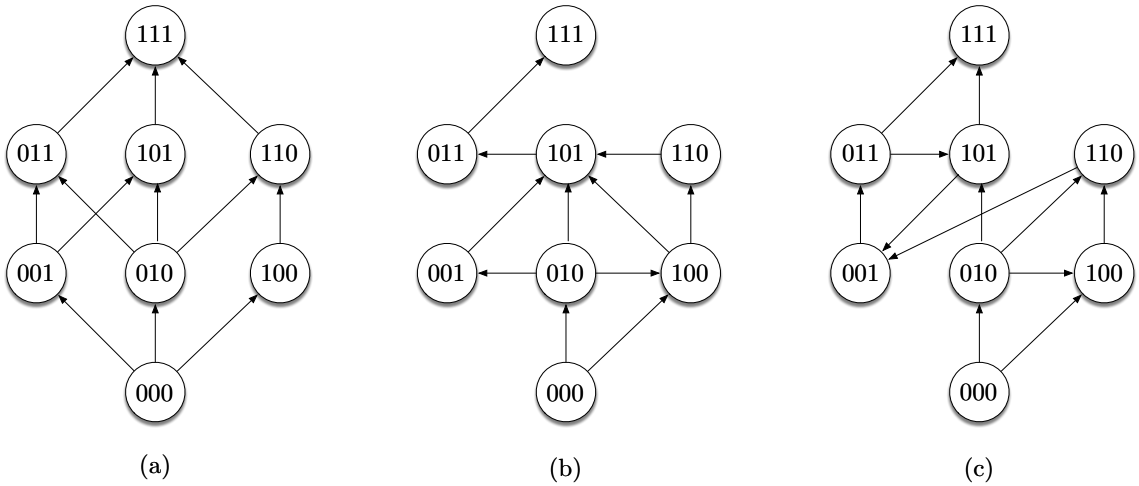


Figure 6.2: The graphs enumerate all possible combinations of test outcomes for three exemplary three-test problems. The edges mark the possible transitions that can be realized by a hypothetical iterative search algorithm.

For a given set of tests T, the set of all possible outcome vectors can be conveniently visualized as a graph with $2^{|T|}$ nodes. Figure 6.2 presents three such graphs, which we will examine in the examples that follow. Each node in the graph is associated with a certain combination of interaction outcomes, and each candidate solution that interacts with the tests in T is assigned to one node based on its outcome vector. Nevertheless, due to many-to-one genotype-phenotype mapping, many of them are assigned to the same node. Also, because in a typical test-based problem the variance of tests' difficulty in T is high [149, 151], some outcome vectors are more likely than others to occur in an evolving population, so that the distribution of candidate solutions over the graph is non-uniform.

The top node in the graph corresponds to the ideal solution that passes all the tests in T and therefore achieves $f_v(s) = |T|$, while the bottom node is identified with the worst solution with fitness $f_v(s) = 0$. The intermediate nodes, on the other hand, represent the whole gamut of possible behaviors (combinations of interaction outcomes) that solutions might exhibit, with nodes on the same level having equal fitness according to f_v.

The arcs in Fig. 6.2 illustrate the possible *transitions* between the outcome vectors that are realizable by a search operator $m : \mathcal{S} \to \mathcal{S}$. A search operator m might be able to enhance a candidate solution with a capability to solve a new, previously unsolved test and thus moving it one level up in the graph. For instance, an arc connecting the node 001 to the node 011 in Fig. 6.2a indicates that there exists at least one pair of candidate solutions (s_1, s_2) such that $m(s_1) = s_2$ and $g(s_1, t_1) = 0$, $g(s_1, t_2) = 0$, $g(s_1, t_3) = 1$, $g(s_2, t_1) = 0$, $g(s_2, t_2) = 1$, and $g(s_2, t_3) = 1$. It is also possible for an operator to alter multiple elements of an outcome vector and so move it several levels up, down or sideways in the graph; however, for simplicity, the figure does contain such examples.

Which of the paths in such graphs will be traversed by an evolving population depends on the interplay between a search operator and an evaluation function. The difficulty of a given problem will in general depend on the probability of reaching the top node.

Example 6.2. Consider a hypothetical problem instance with a transition graph shown in Fig. 6.2a. Assume a search algorithm equipped with a search operator m, which starts with one or more candidate solutions in 000, i.e., such that fail all tests. It does not take long to realize

that such a problem is easy to solve: the transitions are aligned along the gradient of the objective function f_v, so even the simplest hill-climbing algorithm will likely traverse the path from 000 to 111.

The problem in Fig. 6.2b is also solvable, as a path from 000 to 111 exists. Nevertheless, the transition from 110 to 011 is not accompanied by an improvement (f_v remains to be 2). Because the scalar evaluation function imposes a vertically oriented gradient in the graph, a search algorithm that accepts only moves that lead to strict improvements will get stuck in either 110 or 101 on its way to 011. This is not much of a problem for stochastic, parallel search algorithms such as EAs, which could still move from 01 to 10 by pure chance, or find another way to 11 that does not involve 01.

Consider however the problem shown in Fig. 6.2c Once search reaches the combination 110 or 101, further progress can be made only by moving to the combination 001, which implies decreasing f_v from 2 to 1. Only search algorithms that accept such deterioration can escape this trap and so avoid premature convergence. Such graphs are generally harder and more demanding to traverse than those shown in Fig. 6.2a and Fig. 6.2b. ■

The above example is simple for the sake of clarity. In practice, the transitions between combinations of test outcomes, rather than being possible or impossible, will be more or less *likely*. Nevertheless, the problem will persist and manifest in the *likelihood* rather than the *possibility* of reaching an optimum. The conclusion that follows is that scalar evaluation does not reveal behavioral differences between candidate solutions. In particular, candidate solutions in the same layer of a transition graph are treated equally even though their outcome vectors are significantly different. Furthermore, in consequence of search bias and varying difficulty of tests, certain outcome vectors are more likely to be attained than others, making some paths in transition graphs more favorable. In a longer run, if solutions with easy-to-attain outcome vectors dominate the population, the risk of premature convergence raises accordingly.

The presence or absence of various paths in the above graphs is also closely related to fitness landscapes [366]. In particular, the case in Fig. 6.2a can be associated with unimodal fitness landscape, the one in Fig. 6.2b with a *plateau*, and the one in Fig. 6.2c with a *trap* (*deception*). This is however where the analogy ends. Fitness landscapes visualize a scalar objective function and stretch over the space of solutions arranged with respect to the actions of search operators. The nodes in our graphs correspond not to candidate solutions, but to the behavioral equivalence classes determined by combinations of interaction outcomes.

In non-trivial problems, transition graphs will be not only large, but also very 'tangled'. This is because the mapping from the 'genotype' of candidate solutions (the elements of $\mathcal{S}$) to phenotype/behavior (the elements of $\{0,1\}^{|\mathcal{T}|}$) can be particularly complex. A minute modification of the former may cause a dramatic change in the latter. On the other hand, even a major change in genotype can be phenotypically neutral. The domains of game playing and program synthesis are good examples here. In games, the complexity of the genotype-phenotype mapping stems from the usually sequential nature of games, where rewards for players are known only after they have made a series of moves. In program synthesis, this complexity results primarily from the interactions between instructions within a program. In [136], a weighted graph similar to those in Fig. 6.2 was constructed, with nodes corresponding to combinations of outputs of GP programs, and the weights of edges reflecting the likelihood of moving from one behavior to another. The graph was strongly asymmetric, with some transitions very common and some extremely unlikely (see Fig. 2 in [136]). As a consequence, some nodes were almost isolated from the remaining part of the graph, which made them particularly difficult to arrive at.

Last but not least, there are some similarities between transition graphs and Markov Decision Processes (MDPs) [295]. MDP is a mathematical framework for modeling decision making in stochastic environment. At each time step, the process is in some state, and the agent chooses an action that is available in this state. The process (environment) then responds by transitioning into a new state, and giving the agent a reward. State transitions in MDPs satisfy so-called *Markov property*, i.e. the effects of an action taken in a state depend only on that state and not on the prior history. The goal of an agent is to maximize some function of the rewards. It is rather easy to notice that nodes (outcome vectors) in a transition graph may correspond to states in MDP. Arrows that reflect the changes of outcome vectors resulting from modification of candidate solutions could be interpreted as actions taken by the agent in the environment. There is however no direct counterpart of a reward function in a transition graph. The only feedback is the information whether the ideal solution was found at the end of search.

6.5 Chapter summary

In this chapter, we introduced the problem of evaluation bottleneck and discussed its implications that originate in the aggregation of outcomes of multiple interactions between a candidate solution and a set of tests. Evaluation functions that perform such an aggregation are common in practice, particularly in the domain of test-based problems that embraces a wide spectrum of optimization and machine learning problems. Arguably, they form a convenient and minimalistic way of assessing candidate solution's quality, but they also come with a price: compensation of interaction outcomes, loss of gradient, or discreteness are all inherent properties of an aggregative evaluation function that have detrimental consequences on search performance, including crippled generalization and limited scalability. Other, more subtle shortcomings of scalar evaluation, such as unforeseeable search bias that favors only some paths when navigating a search space, come to light once we scrutinize the interplay between the components of a search algorithm, and analyze its dynamics as a whole.

Scalar evaluation obtained by an aggregative evaluation function prevents a detailed insight into a candidate solution's characteristics and incurs inevitable information loss. Solutions that solve equal number of entirely different tests receive the same evaluation and are rendered indistinguishable. Also, such aggregation is negligent to the fact that some tests can be inherently more difficult than others, or more or less harder to reach for a given search algorithm. By agreeing to aggregation of test outcomes into a single number, the conventional search algorithms are oblivious to these aspects and have no insight into the actual, complex interactions taking place between candidate solutions and tests. As a result, they are prone to premature convergence and inferior performance.

It may be worth mentioning that evolutionary algorithms, by performing more or less global parallel search, are *in principle* resistant to premature convergence, because their stochastic nature allows them to visit (albeit only in the limit) all points in the reachable search space. However, from practical point of view, such guarantees are of little use, given finite resources and computational time. Furthermore, EAs find it hard to scale well with task difficulty and the number of tests in T. As larger problem instances are considered, EAs tend to lose efficiency and become less effective at obtaining robust solutions. It seems therefore reasonable to alleviate the bottleneck between the evaluation function and the search algorithm by providing the latter with richer information on solution characteristics and so enabling it to perform better. Doing so is our primary concern in this thesis, and in the following chapters we propose practical methods to achieve this goal.

Another interesting point that emerges from our considerations is that solving particular combination of tests might be critical for a search algorithm to successfully tackle a test-based problem. This observation arises from the analysis of transition graphs (cf. Section 6.4) in which certain outcome vectors are more likely to be attained if candidate solutions already demonstrate some desirable characteristics, meant here as the ability to pass specific combinations of tests. Such capability to solve a subset of tests maybe linked to a *skill* that has to be mastered before other tests can be successfully attained. We anticipate that nontrivial problems may require presence of mutually-exclusive skills, i.e., such that it is difficult to simultaneously make progress on all of them. It maybe therefore important to allow multiple such skills coexist in the population. They may also be a valuable source of knowledge about the problem structure. Indeed, some areas of the search space are often related to subproblems of the original problem, and can be considered as an analog to the concept of minimal coordinate systems [35], in which the axes can be interpreted as the crucial set of skills needed for successfully operating in the given environment. These insights gave rise to some of the proposed algorithms that are discussed in the subsequent chapters.

Chapter 7

Multi-Criteria Evaluation in Test-Based Problems

In the previous chapter, we demonstrated that scalar evaluation affects not only the internal dynamics of search algorithms, but also renders candidate solutions solving different subsets of tests indistinguishable. In the following, we address this problem in the latter context, proposing a means for a many-aspect assessment of solutions produced by algorithms applied to test-based problems. To this end, in Section 7.3 we introduce *performance profile*, a multi-criteria performance evaluation method that characterizes performance using, rather than a scalar, a *vector* of results against tests of various difficulty. Next, in Section 7.4 we introduce two methods of sampling tests for performance profiles, which allows us to obtain robust performance estimates on tests. In Section 7.5, we demonstrate the versatility of performance profiles by applying them to Othello and a variant of the Iterated Prisoner's Dilemma. Last, we carry out a comparative analysis of performance profiles of a well-performing evolved Othello player and a set of players known from past works. The observed differences, which would pass unnoticed or remain unexplained when using scalar performance measures, provide new insights into the characteristics of the considered algorithms. The approach presented in this chapter has been originally published in [149] and later extended in [150, 151].

7.1 Motivation

The challenge in designing effective algorithms for test-based problems (cf. Chapter 5) consists in, among others, obtaining accurate evaluation of solutions. An objective assessment of solution's performance is often computationally too expensive to be useful in practice. For example, expected utility (5.3.1), arguably the most popular performance measure in test-based problems, is the expected score obtained against all tests. When put in game-theoretic terms, it can also be viewed as the expected score of a game playing strategy against a random opponent strategy. However, calculating the exact expected utility even for simple problems is computationally intractable (cf. Section 5.1). Therefore, a common practice is to employ approximate methods of evaluating solution performance, which rely on a limited number of interactions between solutions and tests. The most frequently used evaluation measures are:

- an average score against a pool of fixed, manually-designed tests [226, 93],
- a round-robin tournament between the co-evolving entities [309, 341, 147], or
- an estimated expected utility (the average score obtained against a random sample of tests) [53, 149].

A common feature of all conventional evaluation functions, whether exact or approximate, is that they aggregate the results of multiple interactions into a single scalar value. Though convenient and compact, the performance value obtained in this manner tells very little about the differences between solutions. As discussed in Section 6.2, aggregation of interaction outcomes leads to evaluation bottleneck that severely hinders the progress of search algorithms. One of the consequences of evaluation bottleneck is compensation that results in candidate solutions receiving the same evaluation, even if the outcomes of their interactions with the same tests are completely different. By the same token, aggregative evaluation functions are also oblivious to varying test difficulty. To alleviate this problem, in Section 7.3 we propose to 'multi-objectivize' (term borrowed from [168]) the assessment of candidate solutions and present the underlying information in a structural way. In the following, we define difficulty of a test $t \in T$, the key concept of a performance profile.

7.2 Test difficulty

As already mentioned in Section 6.2, tests in T typically differ in their difficulty. We say that a test is *difficult* if a candidate solution is expected to get a low outcome from an interaction with it; and *vice versa*: it is *easy* if a candidate solution is expected to get a high outcome. In order to formalize test difficulty, we extend the previous definition of interaction function (Def. 5.1) so that it provides two separate outcomes

1. An interaction of a candidate solution s and a test t produces an outcome for s denoted $g_s(t)$, as well as an outcome for t, denoted as $g_t(s)$,
2. The outcomes of interactions fulfill that $g_t(s) + g_s(t) = C$ for all $s \in S$ and $t \in T$, where C is a problem-specific constant.

Without loss of generality, we assume that $C = 1$ and $0 \leq g_t, g_s$, thus $g_t, g_s \leq 1$. For example, if an Othello player s wins against a player t, $g_s(t) = 1$ and $g_t(s) = 0$; when s loses against t, $g_s(t) = 0$ and $g_t(s) = 1$, and $g_s(t) = g_t(s) = 0.5$ in the case of draw.

Definition 7.1. We define the difficulty of a test as the following function $D_{\mathcal{S}} : \mathcal{T} \to \mathbb{R}$:

$$D_{\mathcal{S}}(t) = \mathop{\mathbb{E}}_{s \in \mathcal{S}} [g_t(s)] = \mathop{\mathbb{E}}_{s \in \mathcal{S}} [1 - g_s(t)] \,. \tag{7.2.1}$$

Note that both the quality of a candidate solution $Q_{\mathcal{T}}(s)$ (Eq. 5.3.1) and the difficulty of a test range in $[0, 1]$.

By analogy to $Q_{\mathcal{T}}(s)$, computing $D_{\mathcal{S}}(t)$ is infeasible when the number of candidate solutions in $\mathcal{S}$ is large (which is common in practice), so we approximate it using a finite sample:

$$\hat{D}_S(t) = \frac{1}{|S|} \sum_{s \in S} g_t(s), \tag{7.2.2}$$

where $S \subset \mathcal{S}$ is a (computationally manageable) subset of tests. When S is uniformly drawn at random from $\mathcal{S}$, $\hat{D}_S$ is an unbiased estimator of $D_{\mathcal{S}}$.

Notice that for symmetric problems, where $\mathcal{S} = \mathcal{T}$, every candidate solution is a test, and *vice versa*. In such domains, the higher the quality of a solution, the more difficult it is as a test, i.e., $D_{\mathcal{S}}(t) = Q_{\mathcal{T}}(s)$ for $s = t$. However, performance profiles, introduced in the following section, handle asymmetric problems as well.

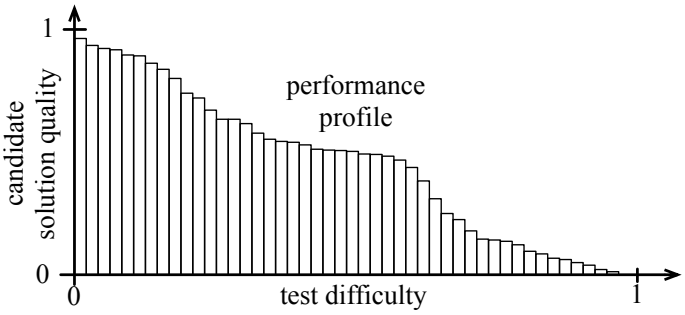


Figure 7.1: An exemplary performance profile. The height of each bar (y-axis) represents the quality of the candidate solution when interacting with tests of difficulty represented by the range occupied by the bar on the x-axis.

7.3 Performance profile

The key idea of the proposed method is *to characterize the performance of a candidate solution as a function of test difficulty*. We will call such a function a *performance profile* of a candidate solution.

Definition 7.2. The performance profile p_s of a candidate solution $s \in \mathcal{S}$ is a function

$$p_s(d) = Q_{\mathcal{T}_d}(s), \text{ for such } d \in [0,1] \text{ that } \mathcal{T}_d \neq \emptyset, \tag{7.3.1}$$

where $\mathcal{T}_d \subset \mathcal{T}$ is the set of all tests of difficulty d, i.e. $\mathcal{T}_d = \{t \in \mathcal{T} : D_{\mathcal{S}}(t) = d\}$. Let us note that $p_s(d)$ is undefined for such ds that there are no tests of difficulty d.

In general, p_s may be uncomputable, because there may be infinitely many difficulty values d for which $p_s(d)$ is defined, and for each such d the set of tests $\mathcal{T}_d$ can be infinite or large. Thus, to estimate p_s, we discretize difficulty into disjoint intervals (*bins*) of equal width. For instance, for 100 bins of width 0.01 a profile becomes a vector of length up to 100.

Definition 7.3. The discretized profile P_s is a function

$$P_s(B) = Q_{\mathcal{T}_B}(s), \text{ for } B \in \mathcal{B}, \text{ such that } \mathcal{T}_B \neq \emptyset, \tag{7.3.2}$$

where B is a bin, $\mathcal{B}$ is the set of bins, and $\mathcal{T}_B \subset \mathcal{T}$ is a set of tests t of difficulty $D(t) \in B$. Notice that P_s is undefined for empty bins.

As in practice $\mathcal{T}_B$ can be too large to compute $Q_{\mathcal{T}_B}$, we fall back to its approximation $\hat{Q}_{T_B}$ (5.3.2), were T_B is a (computationally manageable) sample of $\mathcal{T}_B$.

Figure 7.1 presents the performance profile of an imaginary candidate solution. Each bar represents the performance for a separate bin. We expect typical performance profiles to be weakly decreasing functions of test difficulty, since usually it is easier to get higher payoff from interactions with easier tests than from the more difficult ones. However, there are no fundamental reasons that would prevent a performance profile to take on an arbitrary shape.

Let us notice that each bin B gives rise to a separate performance criterion, and $P_s(B)$ is the quality of a candidate solution s on that criterion. In other words, a bin determines one dimension of candidate solution characteristics. In this sense, performance profile can be considered as a *multi-criteria performance measure*.

In a related work, [9] presented agent-case embedding, which could be also used for characterizing and comparing the performance of solutions of a test-based problem. In contrast to

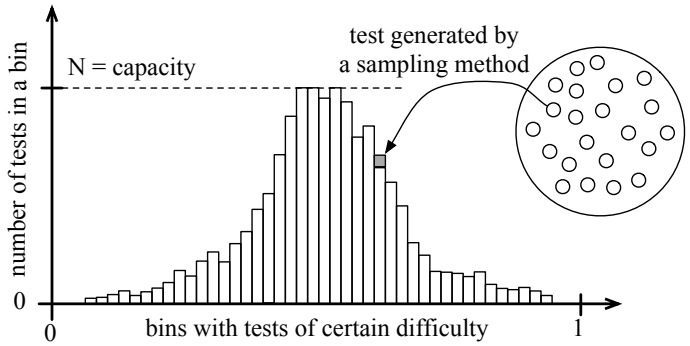


Figure 7.2: A visual illustration of a random sampling method for generating tests for the bins.

performance profiles, which provide a multi-objective solution evaluation on tests of increasing difficulty, agent-case embeddings measure diversity of evolved phenotypes and visualize them in Euclidean space.

7.4 Test sampling methods

The fidelity of a discretized performance profile (Eq. 7.3.2) with respect to its exact counterpart (Eq. 7.3.1) depends on the characteristics of the test samples supporting particular bins. Ideally, every bin should contain the same number of tests, and the tests in each bin should be generated independently.

We propose two methods for generating test samples for bins: *random sampling* and *evolutionary sampling*. Both methods attempt to fill every bin up to *bin capacity* N ($|T_B| = N$ for $B \in \mathcal{B}$). This can be computationally demanding, especially when both N and the number of bins are high. Nevertheless, this is a one-off process: once the bins have been filled up, they can be used *ad infinitum* to assess the performance profiles of arbitrary many candidate solutions.

7.4.1 Random sampling

In random sampling [149], we fill the bins with tests via repetitive independent sampling. In each iteration, we draw at random a test t from the set of all tests $\mathcal{T}$, estimate its difficulty, and place it in the appropriate bin if that bin's capacity has not yet been reached; otherwise, the test is discarded. The difficulty of a test is estimated using M candidate solutions drawn at random from $\mathcal{S}$, independently for every evaluated test (see Fig. 7.2).

The advantage of this method is that it guarantees the independence of tests, within every bin as well as across bins. We thus call the bins generated in this way *unbiased*.

Unfortunately, random sampling does not scale well with bin capacity N and the number of bins. Typically, some bins can be easily filled up, but filling up others becomes very time consuming. For example, in Othello it is difficult to draw very weak or very strong players at random (which are, at the same time, very easy or very difficult tests, respectively). We can expect to run into a similar problem for most nontrivial symmetric test-based problems: a problem for which a very good candidate solution can be easily generated *at random* is simple. This encouraged us to design a more sophisticated evolutionary sampling technique that provides a more balanced bin occupancy.

Algorithm 4 Evolutionary sampling.

```
1:  function EvolSampling(n_samples_fit, n_samples_diff, N, B, pop_size, gen_max)
2:      for B ∈ B do
3:          T_B ← ∅
4:      S ← SampleSolutions(n_samples_diff)
5:                                                  ▷ for difficulty estimation
6:      while not stopped do
7:          F ← SampleSolutions(n_samples_fit)      ▷ for fitness function
8:          P ← SampleTests(pop_size)               ▷ initial population
9:          for gen ← 1, gen_max do
10:             P ← Evolve-Next-Generation(P, D̂_F)
11:             t ← argmax_{x∈P} D̂_F(x)
12:             d ← D̂_S(t)
13:             B ← B' ∈ B : d ∈ B'
14:             if |T_B| < N then
15:                 T_B ← T_B ∪ {t}
16:                 break
17:     return {T_B}_{B∈B}
```

7.4.2 Evolutionary sampling

In the face of challenges troubling the random sampling algorithm, we propose *evolutionary sampling*. In this method, we run an evolutionary process that evolves a population of tests, where test's fitness is defined as its difficulty, approximated using a small number of candidate solutions ($n\_samples_{fit} = 200$). In every generation, we pick the fittest test and calculate a more accurate estimate of its difficulty using a larger number of candidate solutions (e.g., $n\_samples_{diff} = 1\,000$). If this estimate matches a bin that has not yet reached its capacity, the test is placed in that bin and the evolutionary process is stopped. The pseudocode of this procedure is shown in Algorithm 4.

The main advantage of evolutionary sampling is its capability to generate tests of extreme values of difficulty, i.e., very difficult and very easy ones. To provide for both, we run two types of evolutionary processes. In the first type, fitness is defined as test difficulty, so evolution is driven to produce tests of increasing difficulty in consecutive generations. In the second type, fitness is the *negated* difficulty of a test, thus evolution tends to produce the easy tests (note that the extremely easy tests may be as rare as the extremely difficult ones). Compared to random sampling, evolutionary sampling is more likely to fill the extreme bins (the far-left and the far-right ones) up to the desired capacity N. From practical perspective, having well-populated difficult bins is usually more important, as this part of performance profile provides information on how a given candidate solution copes with the most challenging tests.

On the downside, evolutionary sampling is biased. Although each test is a result of an independent evolutionary run, the underlying evolutionary processes may favor certain parts of the test space. However, in Section 7.8 we show that, at least for Othello, the bias is in practice negligible.

7.5 Experimental evaluation

In Sections 7.6 and 7.7, we use performance profiles to characterize and compare candidate solutions produced by different flavors of (co)evolutionary algorithms. All algorithms employ the $(\mu + \lambda)$ generational evolution strategy [22] independently to each maintained population. A population is initialized with μ randomly generated individuals (candidate solutions or tests). In every generation, each of the μ fittest individuals produces λ/μ offspring via mutation (thus, all populations consist of μ parents and λ offspring of those parents). For all experiments, we set

$\mu = 25$ and $\lambda = 25$. In the following, we describe three one-population and two two-population (co)evolutionary algorithms considered in this experiment:

- **Evolutionary Learning with Random Sampling** (EVOL-RS, Fig. 7.3a) is a variant of evolutionary algorithm in which candidate solutions are evaluated against an external set of random opponents T drawn at random from $\mathcal{T}$ once per generation. In order to maintain the same number of interactions per generation as in the other methods described in the following, we set $|T| = \mu+\lambda = 50$. EVOL-RS was shown to surpass one-population coevolution on generalization performance for 1-ply Othello and Iterated Prisoner's Dilemma [53].
- **One-Population Coevolution** (1-COEV, Fig. 7.3a) is a one-population coevolutionary algorithm. All candidate solutions in population interact with each other (in a round-robin tournament). Formally then, the sample T is simply the current population.
- **One-Population Coevolution with Random Sampling** (1-COEV-RS, Fig. 7.3a) is a hybrid of 1-COEV and EVOL-RS that combines the competitive fitness with random sampling. Technically, the sample T is filled in half by candidate solutions drawn uniformly from the current population, and in half by the tests drawn at random from $\mathcal{T}$.
- **Two-Population Coevolution** (2-COEV, Fig. 7.3b) is a two-population competitive coevolutionary algorithm, where individuals are bred in two separate populations, one for the candidate solutions and one for the tests. The population of tests employs $(\mu + \lambda)$ evolutionary strategy, where $\mu = 25$ and $\lambda = 25$. The fitness of a candidate solution s is $\hat{Q}_T(s)$ with the sample T being the current population of tests. Conversely, the fitness of a test t is $\hat{D}_S(t)$ with S being the current population of candidate solutions.
- **Two-Population Coevolution with Random Sampling** (2-COEV-RS), Fig. 7.3b) is 2-COEV hybridized with random sampling. The fitness of a candidate solution s is $\hat{Q}_T(s)$ with T filled in half by the tests from the current population of tests, and in half by the tests generated at random. Compared to 2-COEV, the population of tests in this method is half the size of the population of candidate solutions. The fitness of tests is assessed as in 2-COEV.

Let us stress that the algorithms vary only in the way they assign fitness to individuals, which is illustrated in Figs. 7.3a and 7.3b.

7.6 Experimental analysis of the Iterated Prisoner Dilemma

In this section, we apply the algorithms presented in Section 7.5 to the Iterated Prisoner's Dilemma (Section 5.4.3) and compare the resulting strategies using the single-objective expected utility and the performance profiles.

7.6.1 Experimental setup

In the experiments, we focus on the IPD with $n = 9$ choices (levels of cooperation), which we found to be much more demanding than the 3-choice IPD used in earlier coevolutionary investigations by [53].

Different strategy representations for coevolutionary learning of IPD such as finite state machines [92], and neural networks [62] have been studied in the past. Following [53, 51], we adopt here the arguably simplest one, the direct look-up table [13], and make the players remember the moves from the previous iteration only (*memory-one* IPD). In that case, the n-choice IPD strategy is an $n \times n$ matrix M, where m_{ij} for $i, j = 1, 2, \ldots, n$ specifies the choice to be made given the player's own previous move i and the opponent's previous move j. The other element

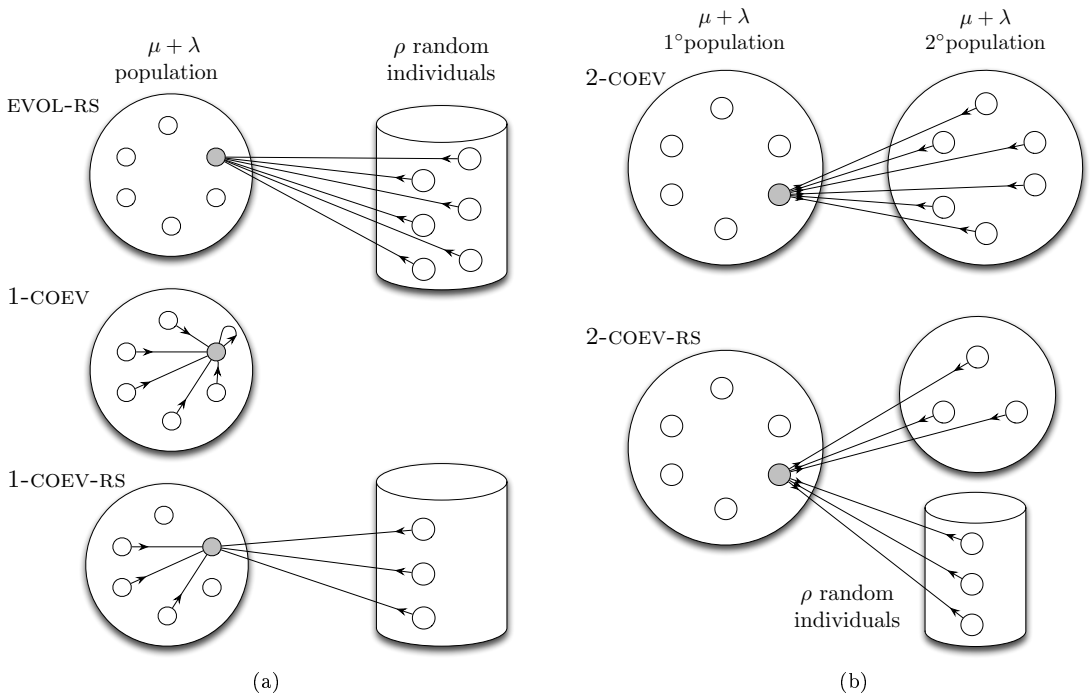


Figure 7.3: The visualization emphasizes differences in fitness assignment among methods considered in this chapter. An arrow means that a game is played between two players.

of the strategy is the initial move m_{00}, which does not depend on the opponent's strategy. A complete IPD strategy is thus determined by $n^2 + 1 = 82$ parameters. The size of search space is $9^{82} \approx 1.77 \times 10^{78}$.

Although the IPD is primarily used to study cooperation [13], we consider it here, following recent works by [50] and [53], as a competitive domain.

Each IPD game consists of 150 PD episodes. To assess the result of an interaction of two IPD strategies we compute their cumulative payoff over the episodes. The highest cumulative score indicates the winner, which is assigned the interaction payoff 1, while the loser gets 0. In the case of draw, the interaction results in 0.5 points for both strategies.

As discussed in Section 7.5, all algorithms considered here maintain a population of 50 candidate solutions which interact with the same number of tests. As a result, in each generation $50 \times 50 =$ 2,500 IPD games are played. Since each evolutionary run consists of 200 generations, it requires the total effort of 500,000 games.

All methods start with an initial population filled with candidate solutions (strategies) randomly drawn from the space of direct look-up tables. The only search operator used by the algorithms is a simple mutation which iterates over all elements of the look-up table and with probability $p_{mut} = 0.2$ replaces the original choice with one of the remaining $n-1$ choices, selected at random. This operator has been found to provide sufficient variation of strategy behaviors for an IPD game with multiple choices [51].

Some of our coevolutionary algorithms and performance assessment methods employ *random players*. Every random player is obtained independently by filling the look-up table with random choices. In the following, by 'random player/opponent' we mean a player obtained in this way. Note that this definition of random player differs from the one that assumes selecting each action by uniformly drawing it from a set of all available actions in a given position. It is, however, coherent with the expected utility measure defined on the set of all tests (see Section 7.3). A

Table 7.1: Expected utilities and 95% confidence intervals of best-of-run individuals obtained by five algorithms for the Iterated Prisoner's Dilemma.

Algorithm	Expected utility
1-CoEv-RS	0.9832 ± 0.0035
Evol-RS	0.9676 ± 0.0042
2-CoEv-RS	0.9561 ± 0.0085
1-CoEv	0.9263 ± 0.0126
2-CoEv	0.9091 ± 0.0131

random opponent is a test drawn a random from the set $\mathcal{T}$. Having said that, the performances of the random players obtained in both ways is similar.

We performed 120 runs for each method presented in Section 7.5. In the following, the best-of-generation candidate solution is the individual with the highest fitness in the population of candidate solutions at that generation (where fitness is subjective and specific for a given method; see Section 7.5). By the best-of-run solution we mean the best-of-generation player of the last generation. In the following, we analyze those players using expected utility (Section 7.6.2) and performance profiles (Section 7.6.3).

7.6.2 Results for expected utility

To estimate the expected utility of an individual (best-of-generation or best-of-run), we let it play 50,000 games against random players. With 1 point awarded for winning the game, 0 for losing, and 0.5 for a draw, the expected utility of a player ranges in $[0, 1]$. In this section, the term 'performance' refers to this measure.

Table 7.1 presents the average performance of the best-of-run individuals for each algorithm accompanied by 95% confidence intervals.

To compare the algorithms, we performed statistical analysis with significance level $\alpha = 0.01$ using the nonparametric Kruskal-Wallis rank sum test, which revealed a statistically significant ($\chi^2 = 116.7$, p-value $< 2.2 \times 10^{-16}$) difference between the results obtained by particular algorithms. A post-hoc analysis using the pairwise Wilcoxon rank sum test with Holm correction indicated the following differences:

$$\text{1-CoEv-RS} > \text{Evol-RS} = \text{2-CoEv-RS} > \text{1-CoEv} = \text{2-CoEv},$$

where '>' denotes significant difference and '=' means no statistical difference.

Let us first discuss the results of 'pure' methods that use homogeneous sets of opponents, i.e., Evol-RS, 1-CoEv, and 2-CoEv. The observed relationship between these methods confirms the previous findings by [53] that evolutionary learning guided by fitness estimates based on random sampling (Evol-RS) achieves higher expected utility when compared to the simple coevolutionary learning approach (1-CoEv). We also observe that two-population coevolution (2-CoEv) is not beneficial in terms of expected utility.

The methods that use a mixture of competitive fitness and random sampling (1-CoEv-RS, 2-CoEv-RS) turn out to be able to evolve strategies with the highest expected utility. There is no statistical difference between 2-CoEv-RS and Evol-RS, but 1-CoEv-RS is clearly the best algorithm for this problem, resulting in the highest median and the lowest variance.

Though the results demonstrate the positive effect of hybridizing different fitness functions, the measure of expected utility does not reveal any details about the strengths or weaknesses of the evolved individuals. For instance, Evol-RS and 2-CoEv-RS produce players of roughly the same

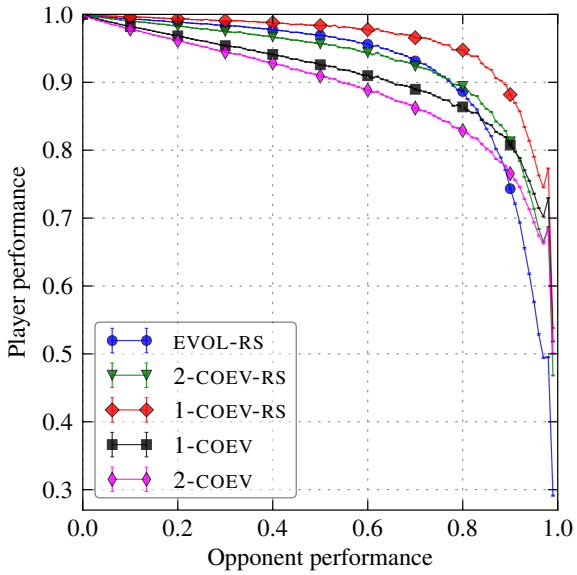


Figure 7.4: Performance profiles for the Iterated Prisoner's Dilemma, for opponent's performance ranging from 0 to 1. Each point (x, y) in a plot indicates the average performance y when playing against the opponents of performance x. This and all subsequent graphs feature 95% confidence intervals, though for many bins they are very narrow.

expected performance, but do they differ in capability of winning with the opponents of particular strengths? In the following we demonstrate how this question can be conveniently answered using performance profiles.

7.6.3 Analysis with performance profiles

In order to scrutinize the best-of-run individuals produced by each algorithm, we apply the performance profiles (cf. Section 7.3). We use evolutionary sampling (cf. Section 7.4) with $(25+25)$-ES ($n\_samples_{fit} = 200$) to generate 100 tests samples, each corresponding to a bin of width 0.01. Each bin's sample is filled up with $N = 5{,}000$ tests, where the difficulty of every test is estimated on the basis of games with $n\_samples_{diff} = 10{,}000$ random players.

Figure 7.4 shows the performance profiles averaged over the best-of-run individuals produced by 120 runs of every algorithm. A point at coordinates (x, y) indicates the average performance y when playing against the opponents of performance x. For instance, the performance of 2-COEV is about 0.9 for the opponents with performance of 0.5 (by which we mean the opponents with performance in the range of $[0.5, 0.51)$, since bin width is 0.01). Recall that the IPD is a symmetric problem, thus player's difficulty (when it acts as a test) is equal to its quality (when it acts as a candidate solution).

The decreasing trend in each profile confirms the supposition that the stronger opponents are harder to defeat than the weaker ones. The only exception of the decreasing trend is the bin $[0.98, 0.99)$ that is 'easier' than bin $[0.97, 0.98)$ for all algorithms. Despite some effort, we were unable to explain this artifact.

Some methods clearly dominate others. The profile of 1-COEV-RS dominates all other profiles, which explains its best result in terms of expected utility (cf. Table 7.1). Also, 1-COEV dominates 2-COEV. The statistical analysis conducted in Section 7.6.2 did not reveal them as significantly different because difficult tests are few and far between in the random sample used to estimate the expected utility. On the contrary, the rightmost bins of performance profiles host many such

Table 7.2: Expected utilities and 95% confidence intervals of best-of-run individuals obtained by five algorithms for Othello.

Algorithm	Exp.Utility
2-CoEv-RS	0.866 ± 0.0024
EvoL-RS	0.8624 ± 0.0023
1-CoEv-RS	0.8371 ± 0.0036
1-CoEv	0.7997 ± 0.0052
2-CoEv	0.7963 ± 0.0064

tests. We take this as evidence that 1-CoEv should be preferred to 2-CoEv for this problem, *even though they perform the same on average.*

Other profiles are mutually non-dominated — their plots cross each other. In this respect, the most interesting is the EvoL-RS profile. Although Table 7.1 suggests no significant differences between 2-CoEv-RS and EvoL-RS, their profiles reveal that 2-CoEv-RS copes with the strong opponents much better, while EvoL-RS is more effective against the weaker ones. Such a profile shape reflects method's trade-off in ability to cope with opponents of various strength. The single-criteria performance measures, like expected utility, are not able to pinpoint such differences and therefore are much less descriptive.

Moreover, the analysis with performance profiles shows that for the strongest opponents (performance > 0.9) EvoL-RS is worse not only than 1-CoEv, but even than 2-CoEv that ranks last on expected utility. For the opponents of the last bin (difficulty 0.97), the expected interaction outcome of EvoL-RS is worse by 0.17–0.26 than for the other algorithms.

7.7 Experimental analysis of 1-ply Othello

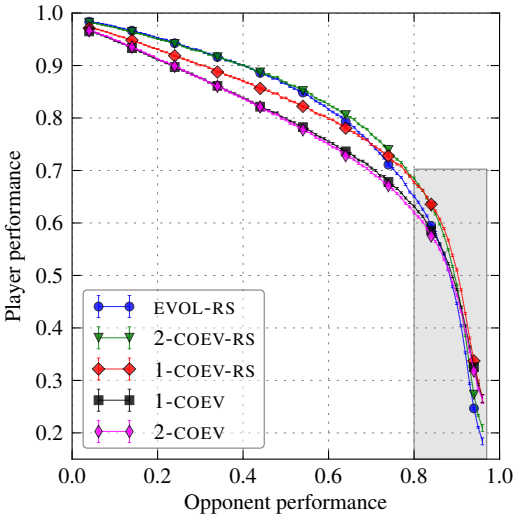
In this section, we apply the five considered algorithms to the game of Othello (Section 5.4.1) and compare the resulting strategies using the single-objective expected utility, performance profiles and a round robin tournament.

7.7.1 Experimental setup

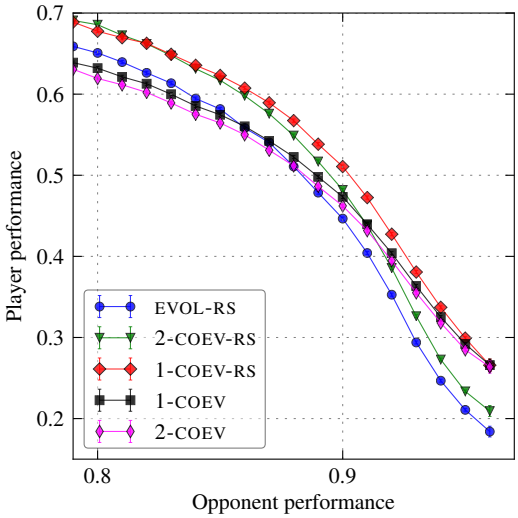
To maintain a similar learning environment to that used for the IPD, we retain most of the evolutionary parameters such as the number of generations, population sizes and the total effort per generation (cf. Section 7.6). In order to learn strategies that are able to play both sides, we 'symmetrize' the game by assuming that a single interaction is a double game, where each of the interacting individuals plays one game as a black player and one game as a white player. With a population of size 50, this leads to 2,500 interactions (double games) or 5,000 single games per generation. In a single game, half a point is divided between the players: the winner receives 0.5 point and the loser 0 points, or they get 0.25 points each in case of a draw. Thus, the result of an interaction is in the $[0, 1]$ range, as it was for IPD.

We represent Othello strategies using position-weighted piece counter (WPC). WPC is a linear weighted board evaluation function which implements the *state evaluator* concept, i.e., it is explicitly used to evaluate how desirable is a given board state. It assigns a weight w_i to a board location i and uses the scalar product to calculate the utility f of a board state $\mathbf{b}$:

$$f(\mathbf{b}) = \sum_{i=1}^{8\times 8} w_i b_i,$$



(a) Performance profiles in 0.0–1.0 range of opponent performance. Grayed region is zoomed on the right.



(b) Performance profiles zoomed to 0.8–0.96 range of opponent performance.

Figure 7.5: Performance profiles for Othello in 0.0–1.0 range of opponent performance. Each point (x, y) in a plot indicates the average performance y when playing against the opponents of performance x.

where b_i is 0 in the case of an empty location, $+1$ if a black piece is present or -1 in the case of a white piece. The players interpret $f(\mathbf{b})$ conversely: the black player prefers the moves leading to the states with a higher value, whereas the lower values are favored by the white player.

We employ WPC as a state evaluator in a 1-ply setup: given the current state of the board, the player generates all legal moves and applies f to the resulting states. The state gauged as the most desirable determines the move to be made. Ties are resolved at random.

The population is initialized with random players whose weights are uniformly drawn from the range $[-0.2, 0.2]$ [226]. The only search operator used by all algorithms is a mutation that perturbs all the weights w_i with an additive noise:

$$w_i^{'} = w_i + 0.1 \cdot U[-1, 1],$$

where $U[-1, 1]$ is a real number drawn uniformly from $[-1, 1]$. Weights resulting from mutation are clamped to the interval $[-10, 10]$. Consequently, the space of strategies is a $[-10, 10]^{64}$ hypercube. As in the case of IPD, we performed 120 runs for each algorithm.

7.7.2 Results for expected utility

We start the performance analysis of the best-of-run solutions by using the scalar measure of expected utility, which we estimate via 25,000 double games (50,000 games in total) against the random WPC players. Table 7.2 reports the results of this experiment.

We performed the same statistical analysis as for the IPD in Section 7.6.2 and obtained the following partial ordering of algorithms:

$$\text{2-COEV-RS} = \text{EVOL-RS} > \text{1-COEV-RS} > \text{1-COEV} = \text{2-COEV},$$

where '$>$' denotes significant difference and '$=$' means no statistical difference at significance level $\alpha = 0.01$.

What this result has in common with the IPD ranking is the superiority of the methods involving random sampling. However, the relationships between them is not the same. In particular, 2-CoEv-RS is now better than 1-CoEv-RS, which is, in turn, worse than EvoL-RS.

7.7.3 Analysis with performance profiles

To generate the samples of tests for bins (each defined as $[x, x + 0.01)$ range of performance), we used evolutionary sampling engaging a $(25 + 25)$ evolutionary strategy with $gen_{max} = 10{,}000$. The fitness and difficulty of each individual has been approximated with double games against $n\_samples_{fit} = 200$ and $n\_samples_{diff} = 1{,}000$ random players, respectively In contrast to the IPD, despite computing on 60 cores of modern CPUs for a few days, we were not able to fill up all the buckets to the assumed capacity of $N = 1{,}000$ opponents. The three first and the three last bins (performance ranges of $[0, 0.03)$ and $[0.97, 1]$) remained empty, and the bins $[0.03, 0.04)$ and $[0.96, 0.97)$ were filled only partially. In total, the 100 samples of tests contain 91,727 opponents of performances ranging in $[0.03, 0.97]$.

Figure 7.5 shows the average performance profiles for the best-of-run Othello players evolved by particular algorithms. In contrast to IPD, it is hard to observe any dominance between the profiles, except for 1-CoEv-RS, which dominates both 1-CoEv and 2-CoEv.

Noteworthy, in the large part of opponent difficulty range (performance of 0.0–0.6), the order of profiles is consistent with the ranking obtained with the single-criteria measure of expected utility. However, the order changes dramatically for the strongest opponents. Strikingly, EvoL-RS and 2-CoEv-RS, the two best algorithms according to the statistical analysis based on expected utility, become the two worst ones when confronted against the strongest opponents (see Fig. 7.5b). In contrast, 1-CoEv and 2-CoEv, the two worst algorithms in terms of expected utility, are significantly better than both EvoL-RS and 2-CoEv-RS on the rightmost bins, showing performance similar to 1-CoEv-RS.

Clearly, the performance profiles reveal the strong points of 1-CoEv, 2-CoEv and 1-CoEv-RS, which pass unnoticed for expected utility. However, attaining higher performance against the stronger opponents is not sufficient to compensate the inferior position when it comes to mediocre opponents, because the latter ones occur much more frequently in an unbiased sample used to estimate the expected utility.

7.7.4 Round-robin tournament

Round-robin tournament (cf. Section 3.5) is a popular method that determines a ranking of methods by playing matches between teams of players they produced [147, 309]. The important conceptual difference with respect to other performance indicators considered in previous sections is the *direct* confrontation between the solutions produced by particular algorithms (rather than referring to an external sample of opponents). In this way, round-robin tournament provides a different means for performance assessment that can be used as an alternative to expected utility.

In our tournament, every team consists of 120 best-of-run players produced by a certain algorithm. Thus, a single match involves $120 \times 120 = 14{,}400$ double games. By 'match score' and 'tournament score' we mean, respectively, team's average score obtained in a single match or in the entire round-robin tournament.

Table 7.3 presents the results of the tournament for the Othello players produced by particular algorithms. By bootstrapping the outcomes of double games, we calculated also the 95% confidence intervals of the scores. We present them for the total score in square brackets. For individual

Table 7.3: The round-robin tournament scores for the five coevolutionary algorithms in Othello. The total scores are followed by 95% confidence intervals.

	Match scores					**Tourn. score [%]**
Algorithm	1-CoEv-RS	2-CoEv-RS	1-CoEv	2-CoEv	Evol-RS	
1-CoEv-RS	–	.522	.521	.526	.556	.531 [.528, .534]
2-CoEv-RS	.478	–	.509	.518	.536	.510 [.508, .513]
1-CoEv	.479	.491	–	.507	.513	.497 [.494, .500]
2-CoEv	.474	.482	.493	–	.5	.487 [.484, .490]
Evol-RS	.444	.464	.487	.5	–	.474 [.471, .477]

matches between methods, the confidence intervals were consistently very close to $[x - 0.006; x + 0.006]$, so they have been omitted.

Evolutionary learning with random sampling loses to all other algorithms in head-to-head matches and its aggregated overall score is significantly lower (confidence intervals do not overlap) than 1-CoEv-RS, 2-CoEv-RS, and 1-CoEv and 2-CoEv. Adding the random sampling component improves both one- and two-population coevolution (1-CoEv-RS vs. 1-CoEv and 2-CoEv-RS vs. 2-CoEv). Also, the one-population variants are consistently better than two-population ones (2-CoEv vs. 1-CoEv and 2-CoEv-RS vs. 1-CoEv-RS). 1-CoEv-RS wins against all the other algorithms and is clearly the best in terms of the round-robin score.

7.7.5 Performance profiles explain round-robin tournament and expected utility

Performance profiles can explain the discrepancy between the rankings based on expected utility and the round-robin tournament. Let us first notice that, contrary to the expected utility assessments which involved random opponents, in the round-robin tournament each individual in a team plays only with the players from the opponent teams, and those players (co)evolved to be strong. According to Table 7.2, the performance of team members ranges between 0.79 and 0.87, well above 0.5, the expected performance of a random player.

Let us analyze the profiles in Fig. 7.5b in this range. The best two algorithms, with performance between 0.79 to 0.87, are 1-CoEv-RS (unquestionably) and 2-CoEv-RS (better than the remaining algorithms on most bins). The other three algorithms are significantly worse but certain differences between them are still observable. In particular, 1-CoEv and 2-CoEv start to surpass Evol-RS when test difficulty reaches ~ 0.87. 1-CoEv is subtly, but consistently better than 2-CoEv in this test difficulty interval. This order is consistent with the ranking obtained in the round robin tournament.

Now let us use the performance profiles to explain the order of algorithms induced by the expected utility measure presented in Section 7.7.2. In that case, games are played against random players, whose performance is 0.5 in average (a random player is equally likely to win and lose a game against another random player). Moreover, random player's performance is most often close to 0.5 (the standard deviation of random player's performance is 0.28, because well- and bad-performing random players are a few and far in between. Thus, it is not surprising that in Fig. 7.5, for opponents of difficulty 0.5, the order of algorithms is consistent with the ranking obtained for expected utility in Section 7.7.2.

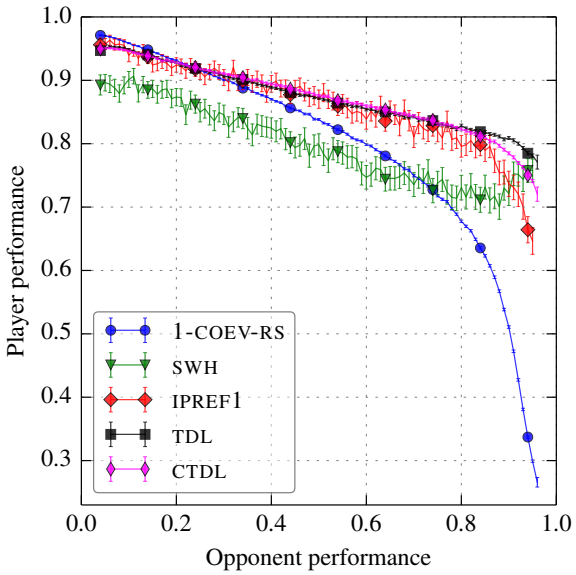


Figure 7.6: Performance profiles of the five players: the handcrafted SWH, and four obtained by the following learning methods: coevolution with random sampling (1-COEV-RS), coevolutionary temporal difference learning (CTDL), temporal difference learning (TDL) or preference learning (IPREF1). Whiskers mark the 95% confidence intervals.

7.7.6 Performance profiles of selected Othello players

Up to this point, our analysis concerned the players produced by (co)evolutionary algorithms. Despite the differences we pointed out, all of the profiles have a similar shape, with high performance for weak opponents that monotonically decreases with the opponents getting stronger. It is thus interesting to verify whether this tendency holds also for other Othello players.

In the following, we analyze the profiles of four players obtained by different methods:

1. Standard WPC Heuristic Player (SWH), hand-crafted by [368], and often used as an opponent in Othello research [226, 340, 237]. Its expected utility is 0.787 ± 0.002.

2. TDL player (0.873 ± 0.003) obtained by [150] using temporal difference learning.

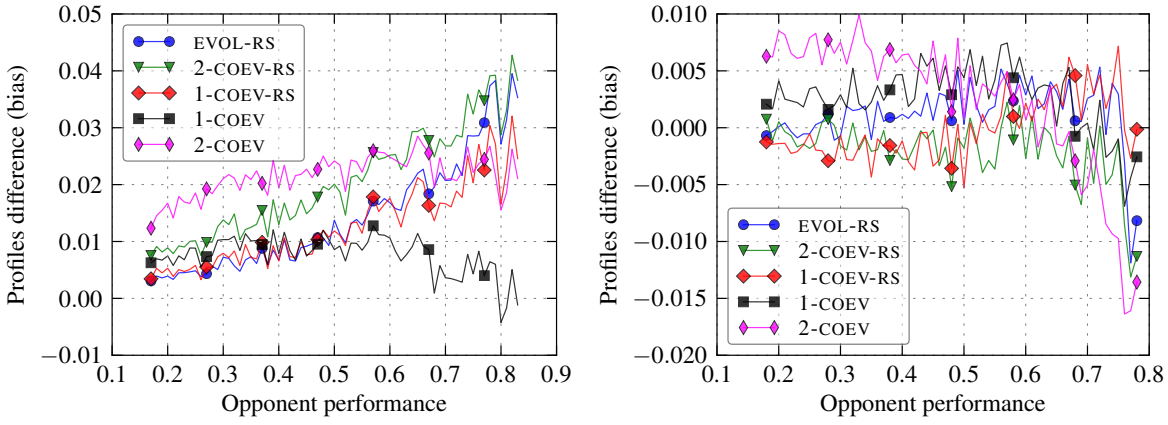
3. CTDL player (0.906 ± 0.001) trained by [338] using a hybridization of coevolution and temporal difference learning [340, 338, 193].

4. IPREF1 player (0.869 ± 0.001) obtained by [306], the only player in this group that represents its strategy using (a simple variant of) n-tuple networks instead of WPC.

Figure 7.6 presents how the profiles of the above players compare to the average profile of 1-COEV-RS (performance 0.837 ± 0.004). Note that the confidence intervals for SWH and IPREF1 are wider because these profiles are based on a single player, while the profiles of 1-COEV-RS, CTDL and TDL are averaged over, respectively, 120 and 30 runs.

Interestingly, no profile strictly dominates all others. CTDL has the highest performance, but, except SWH, the other players are more effective in the left-hand side of the spectrum. Also, the plots show that the flattest profiles characterize the players obtained by the methods that employ temporal difference learning (TDL or CTDL).

The profile of the handcrafted SWH player is significantly different from those obtained by learning algorithms. While the performances of the latter generally decrease with the opponents getting stronger, the SWH player does not strictly subscribe to that trend. Its profile crosses the 1-COEV-RS and IPREF1 curves at about 0.7 and 0.9, respectively, and surpasses them afterwards significantly, even though its overall performance is lower. More surprisingly, the SWH profile seems



(a) IPD (b) Othello

Figure 7.7: The differences between the profiles obtained using random sampling and evolutionary sampling.

then to reverse its trend, and copes better with the strongest opponents (performance of 0.9–0.96) than with the slightly weaker ones (performance of 0.8–0.9).

7.8 The bias of evolutionary sampling

Since the performance profiles used in these experiments rely on evolutionary sampling, which trades better occupancy of extreme bins for bias when generating opponents in bins, it is desirable to quantitatively assess the extent of that bias.

To this aim, we compare the performance profiles obtained using random sampling with the ones generated by means of evolutionary sampling. Random sampling is much worse at generating the strongest and the weakest players than evolutionary sampling. As a result, it fills up only the bins in the interval $[0.17, 0.83]$ for the IPD, and $[0.17, 0.78]$ for Othello. For these non-empty intervals, in Fig. 7.7 we plot the *differences* between the corresponding profiles, which illustrate the extent of the bias of the evolutionary sampling.

Let us notice first that the bias is predominantly positive for IPD, while for Othello it can be either positive or negative. The bias characteristics is thus problem-dependent.

Secondly, the figure shows that the bias of evolutionary sampling is generally low. The maximum absolute difference between profiles is around 0.04 for the IPD, and only 0.015 for Othello. The bias is generally growing with increasing opponent's performance, and can be expected to be higher for the bins to right of 0.8. Nevertheless, the bias is similar for all methods (except for 1-Coev for IPD), thus its influence on the ranking of algorithms is limited. We can, therefore, assume that the bias of evolutionary sampling does not preclude the resulting samples of tests from being reliable and objective performance indicators.

7.9 Chapter summary

In this chapter, we formalized the technique of performance profiles and demonstrated its usefulness for comparing solutions for test-based problems and, implicitly, learning algorithms. Performance profiles proved to be capable of revealing the differences in characteristics between candidate solutions that have similar expected utility. They can also be utilized to explain the discrepancy between the outcomes of a round-robin tournament and expected utility. Because performance

profiles abstract from the internals of learning algorithms, nothing precludes them to be applied to solutions other than best-of-run solutions, and we anticipate that they can be useful for, e.g., explaining the dynamics of search process and characterizing behavior [342] of other (i.e., non-evolutionary) algorithms [150]. In prospect, we envision them as a valuable tool providing researchers with a more detailed feedback on algorithm's characteristics and so helping them design new approaches.

Although we applied here performance profiles to two *symmetric* test-based problems, they are defined in a general way, and thus applicable to the asymmetric case (when $\mathcal{T} \neq \mathcal{S}$) as well. The only formal requirement is that the sum of payoffs obtained in a single interaction by a solution and a test is constant.

For a relatively simple problem such as the IPD, simply drawing samples at random may be sufficient to populate a range of bins that reveal the differences between the algorithms in question. For harder problems like Othello, more sophisticated evolutionary sampling of tests may be necessary to populate the extreme bins and obtain an assessment of the very strong opponents, which may be important in practice. Such instrumentation of profiles proved helpful in this study. However, the more powerful strategy representations, like for instance n-tuples [227], may provide players with performance near 1.0 against all random opponents. In such cases, the specific samples used in our experiment may turn out to be insufficient to reveal any differences, and even more sophisticated sampling methods (or more computational effort) may be required. In any case, we would recommend assessing the bias of the sampling method by confronting (where possible) the bin sample with the random sample, as we did in this chapter.

When it comes to contributions to research in coevolutionary algorithms, we presented evidence that coevolution can offer an advantage over evolution with random sampling for learning game strategies. This advantage is observable for both the IPD and Othello, in the right side of the performance profiles, and in head-to-head comparison for Othello. We conclude thus that an algorithm that autonomously and dynamically redefines its own fitness function (1- and 2-CoEv) can produce strategies which are better in such confrontation than those produced by an algorithm that relies on an stationary (though not static) fitness function (Evol-rs). However, pure coevolution is not sufficient to attain that, and additional means like involvement of random opponents are necessary.

The experiments performed in this chapter may be viewed as a case study that demonstrates evaluation bottleneck (cf. Section 6.1) and its consequences in practice. In particular, we showed that candidate solutions of similar expected quality may indeed specialize in solving different subsets of tests, as illustrated by their performance profiles. In the case of the IPD and Othello, this corresponds to a higher probability of winning the games with opponents of varying strength.

We also gave evidence that the distribution of tests' difficulty in test-based problems tends to be highly non-uniform. When viewed in the context of an aggregative evaluation function, this observation further strengthens our claim that driving the search purely with a scalar performance measure (e.g., the number of solved tests) is inefficient due to inability to differentiate easy and difficult tests during evaluation. It is important to stress here that performance profiles, as introduced in this chapter, are not meant to directly counteract this problem. Rather than that, they are typically used to cast more light on candidate solutions and expose differences in their performance that would otherwise pass unnoticed. Even though they provide a much deeper insight into a candidate solution's quality than a scalar evaluation function, performance profiles cannot be used as alternative means of driving a search algorithm. For this reason, in the following chapters, we focus on the notion of alternative search objectives that can substitute a scalar evaluation function and improve search performance.

Chapter 8

Automatic Discovery of Search Objectives

In Chapter 6, we identified the evaluation bottleneck and argued that the information lost in aggregation of interaction outcomes can be essential for the success of an evolutionary search process. In the previous chapter, we demonstrated one of its practical consequences — candidate solutions with the same or similar expected utility tend to have significantly different performance profiles, meaning that their behavior on the same set of tests is actually entirely different. Although performance profiles proved to be a great analytical tool, they do not address the problem of evaluation bottleneck. In this chapter, we propose the framework for discovery of search objectives in test-based problems, intended to widen this bottleneck by providing search algorithms with richer information on solutions' characteristics. We demonstrate how such behavioral characteristics can be embedded in an algorithm to perform a better-informed and directed search.

We begin our discussion by providing further motivations for the proposed framework in Section 8.1. In Section 8.2, we introduce interaction matrices, and subsequently discuss several possible directions in which these matrices can be exploited to benefit search algorithms. In Section 8.4, we formalize the proposed framework for heuristic discovery of search objectives, describe its conceptual underpinnings, and demonstrate how it may facilitate the design of alternative means of driving a search algorithm (evaluation functions). Afterwards, in Section 8.5, we discuss how search objectives can be employed as basis for selection operators in EAs. We close by reviewing some of the related concepts in Section 8.6. Some of the material presented in this chapter is based on the following publications [214, 213, 186].

8.1 Motivation

The success of evolutionary search algorithms is contingent on two capabilities: generating evolvable candidate solutions, and selecting promising search directions. In evolutionary computation, search operators are responsible for the former, and evaluation and selection mechanisms for the latter. While a large number of search operators have been proposed in the past [252, 275], less attention has been paid to evaluation and selection. On the face of it, limited interest in alternative ways of evaluating and selecting candidate solutions is unsurprising. In the end, many problems approached by various flavors of EAs are formulated as optimization problems, with well-defined objective function (fitness) that in the case of test-based problems boils down to counting the number of passed tests. However, the list of shortcomings of conventional objective functions discussed in Chapter 6 clearly calls for alternative means of characterizing candidate solution's performance. Such means should better inform a search algorithm about other aspects of candidate solution behavior and so broaden the bottleneck of scalar evaluation. In many domains, there

are no principal reasons to conceal the details of evaluation. This is particularly true for test-based problems, where an act of evaluating a candidate solution involves interaction with *multiple* tests and produces detailed information that can be potentially exploited and benefit the search. Our main motivation is therefore to widen the evaluation bottleneck by providing search algorithms with richer information on solutions' characteristics that can be embedded in a running algorithm. In this chapter, we present the framework for *heuristic discovery of search objectives* that aims at providing alternative multifaceted characterizations of candidate solutions. The key concept behind the algorithms that subscribe to this framework is an *interaction matrix*, which holds detailed account on interaction outcomes with individual tests. By embracing the entirety of information available in such matrices, it is possible to automatically devise compact, multi-aspect evaluation of candidate solutions that may serve as a basis for selecting the most promising solutions from the current population.

8.2 Interaction matrix

Recall from Section 5.1 that evaluation in test-based problems can be phrased as candidate solutions engaging in interactions with tests. The outcomes of these interactions depend on an interaction function which is an indicator function of the set of tests passed by a solution s. For instance, given a set of tests T where each test $(x_i, y_i) \in T$ is a pair composed of the input x and the corresponding desired output y, the interaction function g can be characterized as:

$$g(s,t) = g(s,(x,y)) = [s(x) = y],$$

where $s(x)$ is the output produced by s for x, and $[\cdot]$ is the Iverson bracket, i.e.,

$$[x] = \begin{cases} 1 & \text{if x is true} \\ 0 & \text{otherwise.} \end{cases}$$

In a more general setting, if a solution s *passes* a test t according to the interaction function g, then the outcome of their interaction is $g(s,t) = 1$. Otherwise, we say that s *fails* t and $g(s,t) = 0$. We assume that interaction outcomes are binary, though in general various *degrees* of passing tests could be considered. For instance, passing a test could be graded according to the similarity of the actual and desired output, i.e., $g(s,t) = |s(x) - y|$. See also Section 5.5 for more details regarding the interaction function employed in CoEAs and GP.

In the context of the current population P, the outcomes of interactions of all candidate solutions in a given population S with all tests from T can be conveniently gathered in an *interaction matrix*

$$G = [g_{ij} = g(s_i, t_j) : s_i \in P, t_j \in T]. \tag{8.2.1}$$

For a population of m candidate solutions and a set T of n tests, G is an $m \times n$ matrix where g_{ij} is the outcome of interaction between ith candidate solution s and jth test t_j. When interaction outcomes are binary, $g_{ij} = [s_i(x_j) = y_j]$; otherwise, $g_{ij} = |s_i(x_j) - y_j|$.

Example 8.1. Consider a population of tree candidate solutions $S = \{s_1, s_2, s_3\}$ and a set of four tests $T = \{t_1, t_2, t_3, t_4\}$. In order to evaluate candidate solutions, they have to interact with every test in T. Assume that the outcomes of these interactions are as follows: s_1 solves every test, s_2

solves t_1 and t_2, while s_3 solves t_3 and t_4. The interaction matrix G looks then as follows:

$$G = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array}. \tag{8.2.2}$$

The fitness (5.3.2) of a given candidate solution is the sum of the corresponding row of G. For instance, $f_T(s_1) = 4$ and $f_T(s_2) = 2$. Notice that individual rows of G are in fact outcome vectors (6.4.1) that characterize the behavior of a candidate solution. ■

Even for relatively small population size $m = |S|$ and a moderate number of tests $n = |T|$, the $m \times n$ interaction matrix G may become quite large. However, the elements of G are routinely computed almost in any flavor of EAs, as evidenced by (4.4.2) and (4.4.3).

An interaction matrix enables simultaneous access to outcome vectors of all candidate solutions and is a convenient tool for capturing the diversity of behaviors in an evolving population. Crucially for the following considerations, it allows to compare and contrast not only the behaviors of candidate solutions, but also the characteristics of tests, which is one of the key ideas behind the algorithms that derive search objectives. How exactly a search algorithm may benefit from scrutinizing interaction matrices will become clear in the following sections.

8.3 Searching for structure in interaction matrices

Evolutionary algorithms search the problem space by selecting individuals in such a way that better individuals have higher chances of becoming a parent to the next generation. To perform selection of candidate solutions in S, i.e., to determine a subset $S' \subset S$ of *promising* candidate solutions, an evolutionary algorithm has to elicit (implicitly or explicitly) information from an interaction matrix G.

The arguably simplest approach is to aggregate the outcomes of interactions over all tests available in T (rows in G) and adopt the resulting quantity as solutions' fitness that governs the ordinary selection stage (e.g., tournament selection). The advantage of this approach is that f_T is an estimator of expected utility, which is the most common external search objective, so an algorithm is driven by a measure that is consistent with the ultimate goal of search. As discussed in Chapter 6, scalar evaluation incurs information loss, and pairs of solutions that were originally incomparable can receive similar, if not identical, evaluation. The latter case is particularly likely if the underlying interaction function assumes only a few values, which is common in test-based problems (where it is most often two-valued).

An interaction matrix, on the other hand, conveys more detailed information on the structure of the current population – for instance, the distribution of fitness. It also reveals a great deal about the characteristics of tests, not least about their difficulty. To some extent, this additional information has been exploited in the past approaches, which we review in the following sections.

8.3.1 Implicit fitness sharing and related methods

Implicit fitness sharing (IFS) [331, 241, 240] was designed to make an evaluation function aware of varying difficulty among the set of tests. Recall from Chapter 7 that test-based problems with uniform distribution of test difficulty are rarely found in practice, as opposed to problems where test difficulty varies considerably. The conventional evaluation function (6.1.1) is oblivious to that fact and grants the same reward for solving every test in T. As argued in Section 6.4, this may

result in premature convergence, as evolutionary search is opportunistic, and the programs in population tend to learn how to pass the easier tests first. In order to encourage a search process to solve the more difficult tests, it would be thus desirable to increase the rewards for solving them. In theory, this could be achieved using the objective or subjective difficulty discussed in Section 6.2, but estimating either of them requires running a sample of programs, and thus incurs additional computational cost. IFS assesses the difficulty of particular tests based on the current population and *weighs* the rewards granted for solving them. Given a set of tests T, the IFS fitness of a candidate solution s in the context of a population S is defined as:

$$f_{IFS}(s) = \sum_{t \in T:\, g(s,t)=1} \frac{1}{|S(t)|} \tag{8.3.1}$$

where $S(t)$ is the subset of candidate solutions in S that solve test t, i.e.:

$$S(t) = \{s \in S :\, g(s,t) = 1\}.$$

IFS treats thus tests as limited resources: programs *share* the rewards for solving particular tests, each of which can vary from $\frac{1}{|P|}$ to 1 inclusive. Higher rewards are provided for solving tests that are rarely solved by population members (small $S(t)$), while importance of tests that are easy (large $S(t)$) is diminished.

The assessed difficulties of tests change as S evolves, which can help escaping local optima. Fitness sharing can be perceived as a simple form of coevolution, where individuals compete for tests and their fate depends on the performance of other individuals (though there are no direct, face-to-face interactions between individuals). From yet another perspective, fitness sharing is a diversity maintenance technique: an individual that solves a low number of tests can still survive if its competence is rare. In this way, IFS helps reducing premature convergence; it shares this objective with explicit fitness sharing proposed in [115], where population diversity is enforced by monitoring genotypic or phenotypic distances between individuals.

8.3.2 Pareto-coevolution

One could also resort to a more direct approach to perform selection based on interaction matrices, by employing the *dominance relation* $\succ_G$ defined on G (6.2.1). This idea has been intensely used in coevolutionary algorithms, where it gave rise to *Pareto-coevolution* [86, 257] that abandons aggregation of interaction outcomes in favor of using each test as a separate objective. In this way, Pareto-coevolution treats a test-based problem as a multi-objective optimization problem, with as many objectives as there are tests in T. It compares candidate solutions with $\succ_G$ — an individual s_1 is preferred to individual s_2 ($s_1 \succ_G s_2$) if and only if it performs at least as good as s_2 on all objectives, and strictly better on at least one objective.

Fig. 8.1 illustrates an example of Pareto coevolution that involves three candidate solutions and two tests. Tests t_1 and t_2 act as separate objectives, while candidate solutions s_1, s_2 and s_3 are embedded in the space spanned by these objectives, i.e., they are arranged according to their performance on the tests. For instance, s_2 solves both tests and therefore dominates s_1 and s_3, which solve only one test each. Notice, however, that s_1 and s_3 are mutually non-dominated because each of them solves a *different* test.

The dominance relation defined on tests, despite relying on all available and undistorted information on interaction outcomes, is not without its own problems. The number of tests in a test-based problem is typically large (recall, for instance, the number of tests in Tic-Tac-Toe discussed in Section 7.1). In consequence, the likelihood of a solution in S dominating any other is

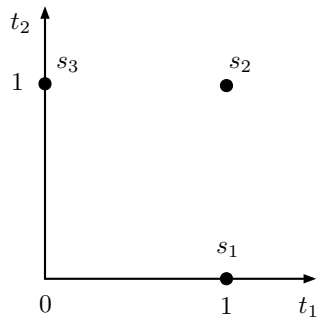


Figure 8.1: An example of Pareto-coevolution, where aggregation of interaction outcomes is abandoned in favor of treating tests t_1 and t_2 as separate objectives, and candidate solutions are s_1, s_2 and s_3 are compared with dominance relation on tests. Candidate solution s_2 dominates the other solutions because it solves both tests, while s_1 and s_3 are mutually non-dominated since each solves a test that the other fails.

rather low, and quickly approaches zero when the number of tests grows. Moreover, it becomes even lower when the tests are diversified, as that increases the chance that each of compared solutions passes a test that the other solution fails. When the dominance relation becomes sparse in this sense, many pairs of candidate solutions are incomparable, making it hard to elicit any useful information that would efficiently drive search.

In a more conventional evolutionary multi-objective optimization setting (i.e., when the objectives are explicitly defined as a part of problem formulation), $\succ_G$ alone also fails to provide a useful search gradient, and algorithms like NSGA-II [75] involved additional mechanisms to perform selection. Nevertheless, even when equipped with such extensions, those algorithms are known to perform well only if the number of objectives is low (typically no more than 3 or 4). Extensions of NSGA-II such as NSGA-III [74] push this boundary, but still perform best if the number of objectives does not exceed 10.

8.3.3 Coordinate systems

Given the limitations of the dominance relation discussed in previous section, it becomes natural to ask whether the number of tests could be reduced, so that the dominance relation may be effectively employed as means for selecting candidate solutions. This question has influenced a substantial body of past research, leading to the hypothesis that originated in the DEMO lab of Jordan Pollack, according to which many test-based problems can be characterized by some form of *internal structure*. As an example, consider a game of chess, where candidate solutions and tests embody different game-playing strategies. In such a setting, candidate solutions could be compared according to their tactics, skills in controlling specific pawns, and other aspects of their play. Each such characteristics may be linked to a *dimension* along which the behavior of candidates could be compared. Knowledge of the problem structure and its dimensions could be used to provide more insight into the nature of test-based problems, and in consequence, to design better algorithms.

Formally, internal structure of a problem consists of a space, often called the structure space, and a function that maps candidate solutions and tests onto that space. The internal structure may be then viewed as a projection of tests onto a smaller set of objectives, such that there exists a one-to-one mapping between the interaction outcomes and the vector of objectives for a given candidate solution. The axes spanning the structure space are often called the *underlying objectives* of a problem. Crucially, such objectives guarantee to preserve all relevant relations between candidate

solutions and tests. The existence of an internal structure of a problem in practice is manifested by the groups of tests that examine the same (or similar) aspect of candidate solutions' performance. The presence of certain patterns in outcomes of interactions that determine behavior of candidate solutions on tests is also a tangible sign of this structure.

The notion of underlying objectives was first introduced in the work on coevolution [35], where it was empirically observed that, under certain circumstances, the tests in a coevolutionary algorithm may identify the objectives that governed the evaluation of candidate solutions. A very similar idea was later presented in the form of an *ideal test set* and *dimension extracting coevolutionary algorithm* [70].

In an effort to make these notions precise, Bucci and de Jong proposed formal methods for framing the internal structure of a problem as a *coordinate system* (CS) [35, 69]. A CS arranges the candidate solutions with respect to axes, each of them being an ordering of tests. A candidate solution is placed in a CS in such a way that it solves all tests below it and fails those directly above it. Consequently, a candidate solution placed high on an axis is preferred to candidates that are lower as it solves more tests associated with that axis. For this reason, an axis in CS is often said to measure some aspect of candidate solutions' quality. If the outcomes of interactions between all candidate solutions and all tests are known for a given test-based problem, a CS can be constructed that exactly reproduces the original dominance relation $\succ_G$, i.e., arranges the candidate solutions so that their spatial relationships in the CS are consistent with $\succ_G$. Interestingly, for every test-based problem there exists a CS of minimal dimensionality [35], and that dimensionality may reflect problem difficulty.

Unfortunately, coordinate systems defined above are exact, which causes them to suffer from several problems. Firstly, by exactly reproducing $\succ_G$, a CS does not provide any additional information that could help driving the search process, despite its ability to compress tests into a lower number of objectives. In particular, if $\succ_G$ is sparse, i.e., few solutions dominate other solutions and there are no grounds for preferring some solutions to the others, even in the space induced by the underlying objectives. The second challenge is the exponential complexity of the algorithms that construct an exact CS from an interaction matrix [144, 141]. In practical terms, this means that a CS cannot be embedded in an algorithm for the benefit of an evolutionary search process. Also, an interaction matrix built from the current populations of candidate solutions and tests is by definition incomplete (because S and T are only transient samples of $\mathcal{S}$ and $\mathcal{T}$), so applying a costly exact algorithm to such a matrix seems particularly wasteful. For these reasons, a CS is an interesting tool for studying the internal structure of test-based problems, but it is not necessarily useful as a means to broaden the evaluation bottleneck and, except for [69], no past work reported using a CS to support a search process.

8.3.4 Other approaches

Other methods that reward solutions for having rare characteristics have been proposed as well. An example is co-solvability [185] that focuses on individual's ability to properly handle *pairs* of fitness cases, and as such can be considered a 'second-order' IFS. Such pairs are treated as elementary competences (skills) for which solutions can be awarded. Lasarczyk *et al.* [203] proposed a method for selection of fitness cases based on a concept similar to co-solvability. The method maintains a weighted graph that spans fitness cases, where the weight of an edge reflects the historical frequency of a pair of tests being solved simultaneously. Fitness cases are then selected based on a sophisticated analysis of that graph.

Last but not least, the relatively recent research on *semantic GP* [188] can be also seen as an attempt to provide search process with richer information of programs' behavioral characteristics. Similarly, pattern-guided GP and behavioral evaluation [192] clearly set similar goals.

8.3.5 Summary

Scalarization and dominance, both employed by most of the approaches reviewed in this section, occupy two extremes in characterization of candidate solution's performance. While these approaches have their merits, both suffer from severe limitations — the former directly contributes to the evaluation bottleneck and ignores the wealth of information offered by G, while the latter results in solutions becoming incomparable when the number of tests is large (which is typically the case in test-based problems). As a result, both fail to provide a useful search gradient in most cases. These observations clearly call for alternative ways of scrutinizing interaction matrices and leveraging individual interaction outcomes for deriving a computationally tractable multi-aspect characterization of candidate solutions. One interesting possibility, which we explore in the proposed framework for discovery of search objectives in the section that follows, is to combine the advantages of the two approaches, i.e., to preserve *some* information on dominance while avoiding aggregation of interaction outcomes into a single scalar value.

8.4 Heuristic Discovery of Search Objectives

8.4.1 Rationale

In the theory of optimization, the *No Free Lunch Theorem* [363] states that all learning algorithms are expected to perform poorly over some problems. One of its implications is that, in order to perform well, a search algorithm should be tailored to a given problem [364]. From another perspective, this may be viewed as an argument for introducing certain *bias* into search methods, and thus allowing a learning system to be shaped towards the salient features of a problem. While this bias can take many forms, it is most often associated with some domain knowledge implemented directly into an algorithm by a human expert to, e.g., set a reasonable starting point for a search algorithm, or to restrict the search space. Douglas Lenat, a prominent researcher in artificial intelligence, once said:

> "*All our experiments in AI research have led us to believe that for automatic programming, the answer lies in knowledge, in adding a collection of expert rules which will guide code synthesis and transformation* [209]."

Even though Lenat refers explicitly to automatic programming, his claim generalizes to other paradigms of computational intelligence, as evidenced by the work of other researchers [244, 279, 249]. The possibility of enhancing a learning process by some sort of heuristic knowledge, like the above mentioned expert rules, has been pursued in the EC community for quite some time now, and a vast body of past research has further built on this idea. One example are so-called *helper objectives* [152] occasionally used in EC and typically designed and/or engaged manually (like, e.g., program size in GP [25]; see also Section 8.6 for an in-depth review of related concepts).

In this thesis, however, we are driven by the prospect of *automatic* discovery of such knowledge. To achieve that goal, the knowledge in question needs to be 'distilled' from some source. Here, we posit that such the interactions between candidate solutions and tests could form such a source, i.e., that certain *behavioral patterns* in the outcomes of interactions between candidate solutions and tests may be present that reflect their skills and shed additional light on their characteristics. Potentially useful behavioral patterns may for instance involve solving specific combinations of

tests or detecting co-occurrence of candidate solutions that have the same or similar outcome vectors. In competitive environments, identifying such patterns may help discovering a diverse and strong set of novel of opponents (tests), worth the computational effort of beating. There are many other behaviors that can characterize both candidates and tests. Crucially, they tend to be accurate indicators of prospective performance, because they reveal meaningful dependencies between candidate solutions, tests and outcomes of their interactions. By making search algorithms capable of detecting such behavioral patterns, we hope to promote behaviorally diverse candidate solutions with the potential to perform well in the future, even if at the moment of evaluation they appear inferior to other evolving candidate solutions.

To further motivate the rationale behind designing methods that search for patterns in behaviors that are relevant for a given test-based problem, let us briefly consider the role pattern recognition plays in biology. There are myriads of ways in which patterns manifest themselves in nature, from the design of a spider's web to the spiral of a nautilus shell. Indeed, our world is full of patterns. Think of how an incoming storm is predicted based on the patterns of warm and cold fronts, where they originate, the directions in which they move and the climate conditions they will confront. Detecting such patterns, rules and regularities in information is a hallmark of natural intelligence. As a matter of fact, being able to recognize patterns is what ultimately gave humans their evolutionary edge over animals. The father of artificial intelligence Marvin Minsky was among the first to extensively explore the link between pattern recognition and human intelligence in order to endow machines with human-like perception and intelligence [247]. His legacy was later continued by his student Ray Kurzweil who claims that all learning results from massive, hierarchical and recursive pattern recognition processes taking place in the brain [196]. As an example, consider how humans learn to read — they recognize the patterns of individual letters first, then the patterns of words, then the groups of words, and eventually gain the ability to read whole paragraphs. Not only are patterns ubiquitous in nature and our life, but they bring along both order and predictability necessary for intelligent behavior.

Clearly, evaluation in test-based problems has the potential of providing extensive information on patterns in behaviors of candidate solutions. Our key observation is that useful behavioral patterns, which we conceptualize in the following as *search objectives*, emerge in an interaction matrix that characterizes an evolving population. By identifying these objectives, we hope to elicit more information on candidate solutions and broaden so the evaluation bottleneck (cf. Section 6.1). The framework we propose in the subsequent sections is intended to lay foundations for algorithms that discover search objectives in a heuristic, computationally tractable manner. We posit that such objectives can be discovered automatically, in a largely data-driven manner, and the information acquired in this way can be used to perform a better-informed and directed search.

8.4.2 Search Objectives

In this section, we introduce the concept of search objective, a formal object designed to encapsulate the problem objectives and guide a search process by creating a useful gradient toward better quality solutions. It originates in the observation that evaluation in test-based problems results in detailed information on behavior of candidate solutions (cf. Section 8.1). A search objective aims at better exploitation of behavioral information in an interaction matrix G, and typically examines only selected aspect(s) of candidate solution's characteristics. When embedded in a workflow of conventional EAs, search objectives facilitate design of alternative evaluation functions that characterize candidate solutions in multidimensional way and thus help to avoid the harmful consequences of evaluation bottleneck.

To better understand this new concept, let us begin the discussion by formally defining search objectives.

Definition 8.2. A search objective for a test-based problem (S, T, g) is a function

$$d : \mathcal{S}^m \to \mathbb{R}^m, \tag{8.4.1}$$

that maps an m-tuple of candidate solutions from S to an m-tuple of their evaluations (scores), where m is the size of population S. The mapping realized by d is based on interactions between elements of S and a subset $T' \subseteq T$ of tests that are performed using an interaction function g.

It follows from the above definition that a search objective is not obliged to meet all the requirements commonly imposed on evaluation functions. In particular, evaluation of a candidate solution s does not have to be independent of other elements of S. The signature given in (8.4.1) allows a search objective to simultaneously evaluate an entire population of candidate solutions. For this reason, it is convenient to think about a search objective d as a vector function

$$d = (h_1, h_2, \ldots, h_m), \tag{8.4.2}$$

where $h_i : S^m \to \mathbb{R}$ is a component function that evaluates a candidate solution s_i based on its performance on T', but also on the performance of candidate solutions $s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_m$. Accordingly, s_i may receive different scores depending on the *context* of evaluation, i.e, the remaining elements of S that were passed as the argument to d. For convenience, we will sometimes abuse the notation and write simply $h_i^d(s_i)$ to denote the score assigned by the search objective d to the ith candidate solution in the context of other members of S.

Despite being oriented at contextual evaluation, the above definition is sufficiently general to embrace also a conventional, one-argument evaluation function $f : \mathcal{S} \to \mathbb{R}$. Functions of this form are commonly employed in EC and can be easily expressed as a context-free search objective

$$d(s_1, \ldots, s_m) = (f(s_1), \ldots, f(s_m)), \tag{8.4.3}$$

where the scores assigned to candidates are independent of each other. For this reason, search objectives can be seen as a superclass of conventional evaluation functions.

Example 8.3. Some tests in test-based problems tend to be easily solvable by most candidate solutions, hence carrying little information regarding solutions' capabilities. In order to shift the attention of a search algorithm to rarely solved yet possibly more important tests, a search objective d could weigh the outcomes of interactions:

$$h_i^d(s_i) = \sum_j w_j(S) g(s_i, t_j).$$

The parameters w_j increase (or decrease) thus the significance of the outcomes resulting from an interaction with the jth test. Notice that w_j depends on a whole population and a single test and in this sense it corresponds to the entire column of G. In practical terms, w_j could be defined manually to, e.g., reflect test difficulty, as in implicit fitness sharing (cf. Section 8.3.1), or it could be discovered automatically (learned) by analyzing the entire G. ■

In test-based problems, a conventional evaluation function typically depends on all tests in T (see e.g. 4.4.2). The concept of search objective is largely driven by the observation that it is not the number of tests, but the particular combination of them that may be critical for a search algorithm to solve a test-based problem. For this reason, in Definition 8.2, we relax the above

dependency and let a search objective operate on a subset of tests $T' \subseteq T$. As argued in Section 6.5, a candidate solutions' ability to solve T' can be likened to a skill. By corresponding to a combination of tests, a search objective is naturally suited to capture such skills and provides the opportunity to make search algorithms more aware about the differences in behavior of candidate solutions.

Identifying and maintaining a diverse set of skills in an evolving population is desirable for yet another reason. As discussed in Section 6.4, some nodes in a transition graph that enumerates all possible combinations of test outcomes can only be accessed once a candidate solution has certain skill(s), i.e., achieves a specific outcome vector that enables such a transition. A search objective allows to explicitly probe for these skills, and in this sense it establishes a more fine-grained evaluation that potentially leads to finding candidate solutions with unique and relevant characteristics. Such candidates may not be the best in terms of objective quality, but instead they may act as the stepping stones that ultimately lead to the optimal solution.

Due to relying only on a handful of tests, a single search objective cannot be expected to work particularly well in isolation. However, as search objectives may reflect different qualities of candidate solutions, it is natural to use many of them. Driving evolution using multiple search objectives that are 'weak' in the above sense can actually be much more efficient; consider for instance ensembles of machine learning classifiers, where multiple weak learners are pooled together (via boosting [315], bagging [32], etc.) to create a 'strong' ensemble classifier.

Let us also note that search objectives are not limited to evaluating candidate solutions on subsets of tests. They may, for instance, depend exclusively on outcomes of candidate solutions' interactions with individual tests in T, i.e.,

$$d(s_1, \ldots, s_m) = (g(s_1, t), \ldots, g(s_m, t)), \tag{8.4.4}$$

as in the dominance relation defined on tests (6.2.1).

Example 8.4. Consider a population of tree candidate solutions $S = \{s_1, s_2, s_3\}$ and a set of four tests $T = \{t_1, t_2, t_3, t_4\}$. The interaction matrix G that gathers outcomes of interactions between S and T is given below:

$$G = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \end{array}. \tag{8.4.5}$$

The conventional evaluation function f_T (5.3.2) applied to population S produces $f_T(s_1) = 3$, $f_T(s_2) = 2$, and $f_T(s_3) = 3$ as the corresponding evaluations. Candidate solutions s_1 and s_3 solve the same number of tests and are thus indiscernible by f_T. Notice, however, that s_3 has the unique behavior compared to other members of S, i.e., it is the only candidate that solves tests t_3 and t_4. In order to take advantage of this observation, a search objective d could evaluate candidate solutions using only t_3 and t_4 and return the sum of corresponding interaction outcomes only on those two tests:

$$d(s_1, s_2, s_3) = (1, 0, 2).$$

According to d, s_3 is now the most desirable candidate solution. By not being bound to all tests, search objectives promote candidate solutions that would otherwise have little chances of surviving selection. Maintaining a skill to solve t_3 and t_4 in the population may pay off in the long run; an ideal candidate solution could be produced by e.g. a crossover operator that exchanges information between s_1 and s_3. ■

In the context of the above example, let us also note that the absolute values returned by a search objective do not actually matter. From the perspective of order-based selection operators such as tournament selection, it is entirely sufficient to make qualitative comparisons: only the *order* of candidate solutions is relevant during selection. In consequence, the values returned by d could be arbitrary as long as the order $s_2 \preceq s_1 \preceq s_3$ is maintained, where $\preceq$ is a preference relation such that $a \preceq b$ means that b is more desirable that a (cf. Section 5.3).

Let us also note that the spectrum of possible search objectives is by no means exhausted by the examples demonstrated in this section. As indicated by (8.4.1), search objectives represent a large class of functions that are capable of expressing virtually any real-valued evaluation function used in the past studies on test-based problems.

It follows from our considerations, that in contrast to conventional evaluation functions, search objectives are not required to objectively assess candidate solutions in the context of given problem. Rather than that, they may be perceived as an elementary source of information, reflecting only selected characteristics of candidate solutions. In this sense, a search objective can be thought of as defining certain features of a solution that are meant to *guide* search by creating a useful gradient toward better performing solutions. This perspective is particularly important given the iterative nature of evolutionary search process, where it is desirable to promote *evolvable* candidate solutions that lend themselves to further development by means of variation operators.

The definition of search objective in (8.4.1) is essentially agnostic about a particular instance of test-based problem. By not being bound to any specific problem or task, search objectives are to promote *problem-independent* behavioral characteristics of candidate solutions that emerge from interaction matrices. This observations brings us to the question of how to derive such objectives, which we purse in the following section.

8.4.3 Deriving Search Objectives

Search objectives offer an alternative means of driving search algorithms, however they are not to be designed explicitly by humans. The key component of the proposed framework are thus algorithms that *automatically* discover search objectives by analyzing an interaction matrix G. Let us briefly recall at this point that for a population of m candidate solutions and n tests, G is an $m \times n$ matrix with elements corresponding to outcomes of interactions between particular elements of S and T. Formally, discovery of search objectives involves transformation of $m \times n$ matrix G into $m \times k$ matrix G'[1], i.e.

$$u(G) = G', \tag{8.4.6}$$

where u is an algorithm responsible for mapping of G into G', and k determines the number of search objectives to be derived from G. In the following, we often refer to the process implemented by u as *derivation* of search objectives. The terms 'discovery' and 'derivation' will be used interchangeably.

As argued in Section 8.3, test-based problems are characterized by an internal structure that implicitly defines several dimensions along which the behavior of candidate solutions can be compared. The existence of this structure is also reflected in G as indicated by groups of tests that examine the same skill of solution's performance as well as the presence of behavioral patterns that can be observed in outcomes of their interactions with tests. The role of u is thus to *heuristically* discover and express this structure (or more precisely the information it conveys) in the matrix G' that results from applying u to G.

[1] To simplify notation, we use the same symbols G and G' to denote the interaction *matrices* and the associated *spaces* of objectives.

The interpretation of rows in G' remains unchanged, i.e., each row coincides with a member of population S. The columns of G', on the other hand, define k derived search objectives that provide alternative characterization of candidate solutions in S in the context of their behavior on tests in T. The value of jth derived objective for a candidate solution s_i corresponding to the ith row of the derived interaction matrix G' amounts to

$$h_i^j(s_i) = g'_{i,j}, \tag{8.4.7}$$

where $g'_{ij} \in G'$. The elements of G' are required to be non-negative continuous values that are to be maximized. Notice that by this token $g'_{ij}s$ already provide more fine-grained evaluation compared to binary information on either passing or failing a test offered by an interaction function g.

Recall from our discussion in Section 8.3.5 that scalarization and dominance can be seen as two extremes in characterization of candidate solution's performance. With search objectives, we hope to explore the middle ground between the two, and for this reason, we typically aim for k to be much smaller than n.

In order to derive search objectives, u employs knowledge discovery algorithms to 'datamine' an interaction matrix. In other words, the entire process relies heavily on inclusion of *learning* in order find the links between candidate solutions and tests based on their behavioral characteristics. Since there is no supervised signal that would guide the derivation process, the set of learning data experienced by u consists merely of observations (outcomes of interactions) describing candidate solutions and tests (rows and columns of G). Discovery of search objectives is therefore the task of *unsupervised* learning, where the objective is to explain the key features of G and model its underlying structure. More precisely, the goal of u is to find such a representation of interaction outcomes that enables multi-objective selection of candidate solutions and, at the same time, preserves as much information about G as possible.

Let us now discuss, how derivation of search objectives can be embedded in the workflow of algorithms solving a test-based problem $(\mathcal{S}, \mathcal{T}, g)$. In essence, evolutionary metaheuristics like CoEAs (cf. Chapter 3) and GP (cf. Chapter 4) search for an ideal candidate solution by repeating the cycle of evaluation, selection and variation (cf. 2.1):

$$\cdots \xrightarrow{v} S_i \times T_i \xrightarrow{g} G_i \xrightarrow[f_T]{v} S_{i+1} \times T_{i+1} \xrightarrow{g} \ldots, \tag{8.4.8}$$

where $S_i \subset \mathcal{S}$ is the population of candidate solutions and $T_i \subseteq \mathcal{T}$ is the set of tests in the ith iteration of search. The interaction function g applied to all pairs from $S \times T$ produces an interaction matrix G_i that forms the basis for evaluating candidate solutions (e.g by means of scalarization as in Eq. 6.1.1). Next, a selection operator uses the outcomes of evaluation phase in order to select the parent solutions for the next generation $(i+1)$. New populations S_{i+1} and T_{i+1} are then created with evolutionary search operators, and the cycle repeats. For simplicity, in (8.4.8), we combine selection and variation together as a single operation v. Also, because T_i changes with time, the above cycle summarizes the operation of conventional CoEA applied to a test-based problem. If we make an additional assumption that the set of test does not change with time, i.e., T_i is held constant in all iterations of search, (8.4.8) implements a broad class of EAs.

To enhance the evolutionary metaheuristics with the capability to discover search objectives, formula (8.4.8) can be extended by introducing an additional step that involves mapping of the outcomes of interactions in G to the space of derived search objectives G':

$$\cdots \xrightarrow{v} S_i \times T_i \xrightarrow{g} G_i \xrightarrow{u} G'_i \xrightarrow[d_1,\ldots,d_k]{v} S_{i+1} \times T_{i+1} \xrightarrow{g} \ldots \tag{8.4.9}$$

The objectives $d_1, \ldots, d_k$ derived by u replace the conventional evaluation function f_T and form a multi-aspect characterization of candidate solutions in S_i that serves as a basis for selecting

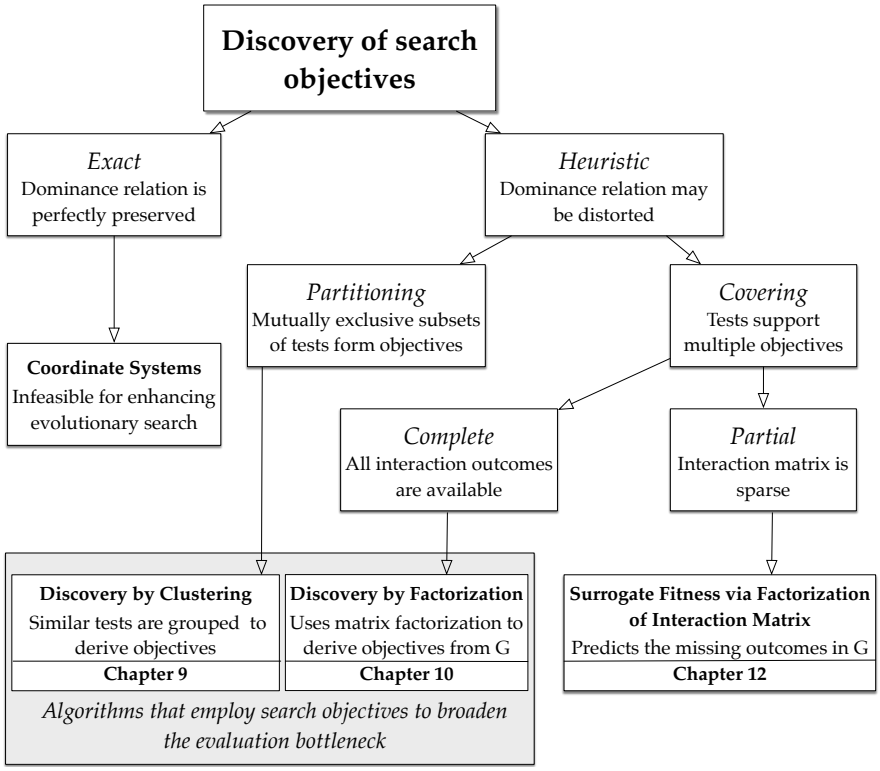


Figure 8.2: The taxonomy of the methods that process an interaction matrix to derive search objectives.

the most promising individuals from S using v. The most commonly used selection operators in EC such as the tournament selection may not readily support evaluation that is not scalar. Rather than adjusting single-objective techniques or devising ad-hoc algorithms for this purpose, it is natural to employ here methods that are inherently multi-objective (e.g. NSGA-II [75]). We discuss this and other possibles of selection under search objectives in Section 8.5.

8.4.4 Taxonomy

To systematize the conceptual approaches to discovery of search objectives and guide our further considerations regarding their properties, we propose the taxonomy shown in Fig. 8.2. The topmost aspect concerns the completeness and precision of the process, i.e. whether u implements an exact or a heuristic approach to derivation of search objectives. Exact methods perfectly preserve the dominance relation in G and are mostly concerned with finding the underlying objectives of a problem (cf. Section 8.3.3), and therefore can be computationally expensive and not necessarily useful as means to drive the search. Heuristic methods that represent the other branch in Fig. 8.2 are more appropriate given the iterative nature of evolutionary search and our desire to derive objectives online, i.e. in parallel to the ongoing search/optimization process. Heuristic algorithms are allowed to distort the original dominance structure in the interaction matrix G. Though this can be considered a downside at first sight, note that even the information in G (which we consider here the primary source of information) does not fully characterize candidate solutions, because of the limited number of tests in T. In other words, it may not make sense to strive to perfectly preserve the information in G, given that that information is anyway incomplete. Moreover, this inconsistency provides us with critical advantages: the resulting dominance relation in G' is more dense and thus likely to impose a reasonably strong search gradient on an evolving population.

Another interesting property of u is its capability to merge several tests into a new derived search objective. In a simplest case, it may be seen as *partitioning* of G into mutually exclusive subsets of tests. In such a case, each test is used exactly once and any pair of original objectives that are mutually redundant (i.e., identical columns in G) are guaranteed to be included in the same derived objective. An example of method that partitions G is Discovery of Objectives by Clustering (cf. Chapter 9) that groups similar tests together with the goal of exploiting these similarities to find a low number of search objectives that characterize candidate solutions in a current population. Otherwise, when the same test may support multiple objectives, u is said to *cover* G. Such a design is more versatile as it allows detecting patterns that overlap columns in G. This allows u to reuse pieces of the interaction matrix for the purpose of deriving multiple objectives. Discovery of Objectives by Factorization, introduced in Chapter 10, is an example of algorithm that implements this approach.

Interestingly, methods that cover G are not required to process every single interaction outcome in G, which opens the door to discovery of search objectives even when G is only partially known (i.e., sparse). Consider for instance problems with an infinite or very large number of tests. Many control problems belong to this category. Even in the discrete domains like Artificial Ant (cf. Section 4.2) or Density Classification Task (cf. Section 5.4.4), the numbers of possible environments (or initial conditions) are often astronomical, not mentioning the continuous domain with problems like inverted pendulum. In such problems, tests (environments) can be generated on demand, and the interaction function performs candidate solution's simulation in an environment and is thus computationally costly. Capability of u to derive search objectives that guide evolution in such conditions can be invaluable. The most obvious way of implementing u when G is sparse is to define objectives using only the available interaction outcomes. Surrogate Fitness via Factorization of Interaction Matrix (cf. Chapter 12) is to a large extent inspired by these observations.

8.4.5 Desired properties

Considering stochastic and iterative nature of evolutionary search, other desirable properties of u involve invariance with respect to:

(i) permutations of rows and columns in G,

(ii) occurrence of multiple copies of candidate solutions and tests in S and T, and

(iii) basic algebraic operations on matrices including scaling (scalar multiplication) and translation (scalar addition).

These properties are also particularly important from the technical perspective. Depending on the flavor of EA, a population of candidate solutions and/or tests is typically implemented as either a set or a list. In the former case, the order of rows and columns in G is undefined and may change (permute) in each iteration of search. In the latter, despite the fixed ordering of elements in lists, there is a risk of introducing duplicates to either population via variation operators. This may potentially lead to many duplicate rows and/or columns in G, hence (i) and (ii) should always be satisfied. At certain point of our discourse the encoding of interaction outcomes becomes particularly important. For this reason, we also wish to guarantee insensitivity of u to scaling and translation (iii) of its elements.

Figure 8.3 illustrates how an interaction matrix G is subject to translation (Fig. 8.3a), duplication of columns (Fig. 8.3b), and permutation of rows and columns (Fig. 8.3c) before the objectives are derived with u. Ideally, u should guarantee that none of these operations affect G'. Recall that search objectives provide evaluation of candidate solutions and help a selection algorithm

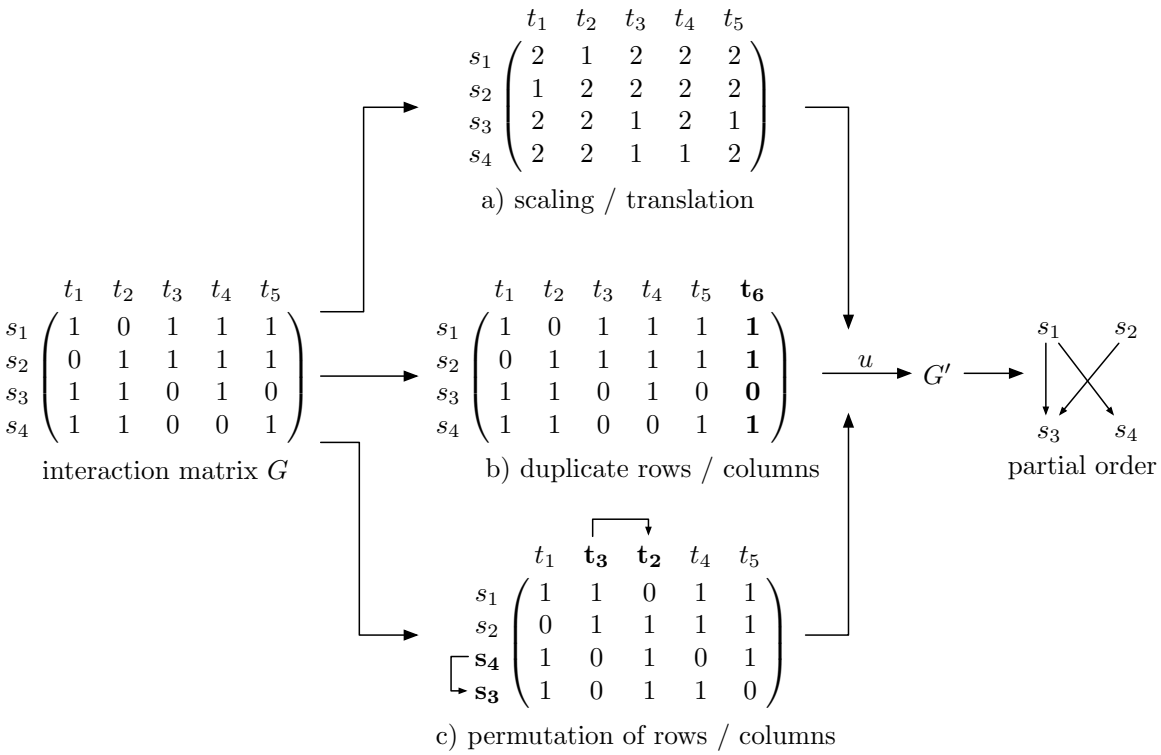


Figure 8.3: Graphical illustration of three desired properties that characterize derivation of search objectives. Matrix G' produced as the outcome of derivation process should maintain the same partial order of candidate solutions regardless of scaling/translation (a), presence of duplicate rows and/or columns (b), and permutation of rows and/or columns in G.

to appoint the best candidate solution within the sample drawn from a population (notice that a given act of selection typically operates only on a subset of individuals rather than the entire population). As mentioned in Section 8.4.2, the absolute values typically do not matter as selection needs only qualitative, ordinal feedback in the form of partial or complete order of candidate solutions. This implies that it is sufficient to require that the order induced by u does not change whenever G is subject to simple transformations discussed earlier.

8.5 Selection under search objectives

From the perspective of selection in evolutionary metaheuristics, the goal of evaluation function is to provide feedback regarding candidate solution's prospective odds of survival. Evaluation therefore serves as the basis for selection operators that decide whether to apply search operators to an individual, or whether to replace or keep it in the population. The framework for discovery of search objectives is designed with a similar goal in mind, however it aims at better exploitation of behavioral information for the sake of improving the communication between evaluation and selection and making the latter aware of subtle differences among candidate solutions. We achieve this is by deriving search objectives from G and characterizing candidate solutions with respect to multiple such objectives. By doing so, we turn a single-objective test-based problem into a multi-objective one. Nevertheless, most of the conventional selection techniques in EC (e.g., tournament selection) are inherently single-objective and cannot be used to appoint the best candidate solution when multiple search objectives are used simultaneously to drive search. In the following, we discuss several alternative means to that end.

8.5.1 Aggregation

The simplest method of dealing with multiple search objectives is to aggregate them into a single, parameterized evaluation function. This approach is commonly employed in the field of decision making [302], but in EC the parameters of such a function, rather than being set manually (e.g., by a decision maker), are sometimes varied during search. The representatives of this class of techniques typically rely on the weighting method [118, 139]. In consequence, candidate solutions are assessed using a particular weight combination (either encoded directly in the candidate or chosen randomly) and all members of the population are evaluated by essentially a different objective function. On the positive side, this leads to simultaneous optimization in multiple directions towards Pareto-optimal front. The potential disadvantage involves a tendency to optimize only the easiest objectives and/or biasing search in a way that favors only convex portions of the front, limiting thus the effectiveness of such algorithms [349].

Since the codomain of search objectives is typically defined on a metric scale, the above approach can be streamlined even further by simply merging search objectives into scalar evaluation using, for instance, arithmetic averaging. Another interesting possibility is to employ the geometric mean, which is equivalent to the concept of *hypervolume* in multi-objective optimization. More specifically, the hypervolume of candidate solution's performance given by the k search objectives $d_1, \ldots, d_k$ can be characterized as

$$h_{vol}(s) = \prod_{i=1}^{k} d_i(s).$$

The key property of hypervolume is that it increases as the scores on $d_i s$ become more balanced. As we showed in [186], this particular form of aggregation proves to be particularly useful in the context of achieving balanced performance on all search objectives simultaneously. As an example, consider two candidate solutions s_1 and s_2 with the same sum of scores on all objectives, i.e., $\sum_i d_i(s_1) = \sum_i d_i(s_2)$. Assume further that the scores of s_1 vary, while those of s_2 are all equal, i.e., $d_i(s_2) = \sum_i d_i(s_2)/k$. In such a case, $h_{vol}(s_2) > h_{vol}(s_1)$.

Scalar aggregation is without doubt a convenient and straightforward approach to handling multiple search objectives. On one hand, it opens the door to employing conventional selection algorithms, but on the other, it significantly increases the risk of evaluation bottleneck (cf. Section 6.1). For this reason, we focus in the following on dedicated multi-objective selection methods.

8.5.2 Switching objectives

Alternative approach to combining the search objectives into a single scalar value is to select candidate solutions according to only one of the k objectives at a time. In essence, this boils down to parallel 'single-objective' search where each selection event is potentially driven by a different search objective. This idea was for instance implemented by Schaffer [314, 313] in vector-evaluated genetic algorithm, where a fraction $1/k$ of each new population is selected using distinct objectives. Fourman [102] proposed a selection scheme in which candidate solutions participate in binary tournaments, randomly choosing one objective to elect a winner of each tournament. These ideas can be further extended, for instance by assigning a probability to each objective that determines the frequency of choosing that objective as a selection criterion during evolution. These probabilities can be either predetermined by the user, chosen randomly, or allowed to evolve with the population [195]. Nevertheless, it has been shown that whenever a fraction of the next parent population is selected using one of the k objectives, there is potential risk of biasing search towards 'mediocre' individuals (i.e. those solutions that do not excel at any particular objective) [299].

Algorithm 5 Lexicase selection.

Input: Population S and set of tests T.
Output: The selected individual.

1. Let $S' \leftarrow S$ and $T' \leftarrow T$.
2. Draw at random an objective $t \in T'$.
3. Set $S' \leftarrow S' \cap c_t(S')$ and $T' \leftarrow T' \setminus \{t\}$.
4. If $|S'| > 1$ and $|T'| > 1$ go to step 2.
5. If $|S'| = 1$, return the only element of S'; otherwise return a randomly selected element in S'.

The bottom line of switching search objectives during evolution is that these objectives tend to act as stepping stones towards the given solution concept. As a result, the evolutionary search may follow more advantageous paths (compared to following just one objective) in the space of candidate solutions and should, in principle, be more likely to find the ideal solution. Despite its appeal, probably the biggest limitation of applying this method in practice is the assumption that a switching criteria which determines when to switch to the next objective has to be defined manually be the experimenter.

8.5.3 Lexicographic ordering and lexicase selection

In order to avoid explicit aggregation, one may also resort to lexicographic approach that provides a total ordering of all candidate solutions without assigning scalar fitness values. In this approach, candidate solutions are ranked by considering each objective in a predetermined order, typically from the most to the least important. In each selection event, two (or possibly more) randomly chosen candidate solutions are first compared on the most important objective. If one of them has superior performance, it is selected. If there is a tie, then the second objective with respect to importance is considered, and so, on until one candidate proves strictly better than the other. In the case that does not happen, a candidate solution is chosen randomly.

Lexicographic ordering has been successfully applied to, for instance, compare individuals under tournament selection in a GA [102]. Another interesting application is by Luke [231], who proposed a straightforward lexicographic parsimony pressure, a multi-objective technique for optimizing both fitness and tree size in GP, by treating fitness as the primary objective and tree size as a secondary objective in a lexicographic ordering.

The lexicographic ordering technique is only useful when the importance of each objective is clearly known *a priori*. This assumption is actually not always met in practice, as it requires a domain-specific knowledge that enables such an ordering. Since lexicographic ordering imposes a total ordering on the space of candidate solutions, it is only appropriate for rank-based selection. Moreover, its applicability is further limited to discrete and coarse-grained objectives. Otherwise, when the objectives are real-valued, it has the tendency to focus almost entirely on the first objective, because candidate solutions will typically have distinct evaluations. This in turn leaves the remaining objectives with hardly any impact on selection. The primary advantages of lexicographic approach include its simplicity and computational efficiency, which makes it competitive with other non-Pareto methods such as the weighted sum of objectives discussed earlier.

Recently, Spector proposed a selection technique based on lexicographic ordering of objectives called *lexicase selection* [333, 127]. A single act of applying lexicase selection to a given population

S under the objectives O proceeds as in Algorithm 5. In each parent selection event, lexicase selection starts by adding all candidate solutions to the selection pool and randomly shuffling the order of test cases. It then proceeds by removing candidate solutions in the selection pool that do not satisfy a *pass condition* c_t on the first test case t. In its original definition, c_t filters all candidate solutions with worse performance than the best performance on the current test t. If more than one candidate solution remains in the pool, the first test case is removed and the procedure is repeated for the next test case until only one candidate solution remains and becomes a parent, or all test cases are used. In such a case, a parent is chosen randomly from the remaining candidate solutions in the pool.

Notice that the entire process is very similar to lexicographic ordering, the main difference being that the adopted ordering of objectives is random and drawn independently in every selection act. This helps avoiding overfocusing on some objectives and diversifies the population. In addition to rewarding the candidate solutions for the best performance on individual objectives, lexicase selection promotes diversified candidates that are clearly superior on a randomly selected subset of objectives. An individual that performs well on objective(s) that are difficult for its competitors has substantial chance to propagate to the next generation even if it performs poorly on many other objectives. For instance, if a single candidate solution alone has the best performance for a particular objective, then it will be selected whenever that objective comes up first in the random ordering, regardless of its performance on other objectives.

Lexicase selection proved to be particularly efficient for so-called *uncompromising problems,* where it is not acceptable for a solution to perform sub-optimally on any objective in exchange for good performance on other objectives [129, 128]. In our studies, we have shown that, in order to improve search performance compared to more conventional selection operators, lexicase selection requires substantial number of objectives to work with; when applied to a moderate number of objectives, its diversification capability and performance deteriorate [218].

8.5.4 Multi-objective selection

Multi-objective evolutionary algorithms (MOEAs, [56, 73]) have become increasingly popular for handling problems with multiple objectives. To a large extent, MOEAs share the same conceptual underpinnings as traditional EAs. The presence of multiple objectives requires however special handling of candidate solutions by a selection operator. Usually, these operators take into account the concept of Pareto optimality and employ the dominance relation (cf. Section 2.2) to rank the individuals in a manner such that the non-dominated individuals have a higher probability of being selected. However, the Pareto dominance only defines a partial order in the objective space, and in consequence, not all candidate solutions can be compared to each other (or, more precisely, the result of their comparison is inconclusive). One way to deal with this limitation is map that partial order to a complete order by partitioning a population into several non-domination classes. Candidate solutions are then ranked based on class membership; those in the same class have equal rank and solutions with a lower rank are preferred. To differentiate the individuals within a class, they are assigned a parameter called *density,* which is intended as a diversity preserving mechanism. A complete order $\prec_c$ is then defined as follows:

$$x \prec_c y \iff (x_{rank} < y_{rank}) \vee (x_{rank} = y_{rank} \wedge x_{density} < y_{density}).$$

The above approach is implemented by, among others, the Non-dominated Selection Genetic Algorithm (NSGA-II, [75]). In order to select candidate solutions for the next generation, NSGA-II employs a tournament selection on Pareto ranks. As a density measure, it uses crowding that

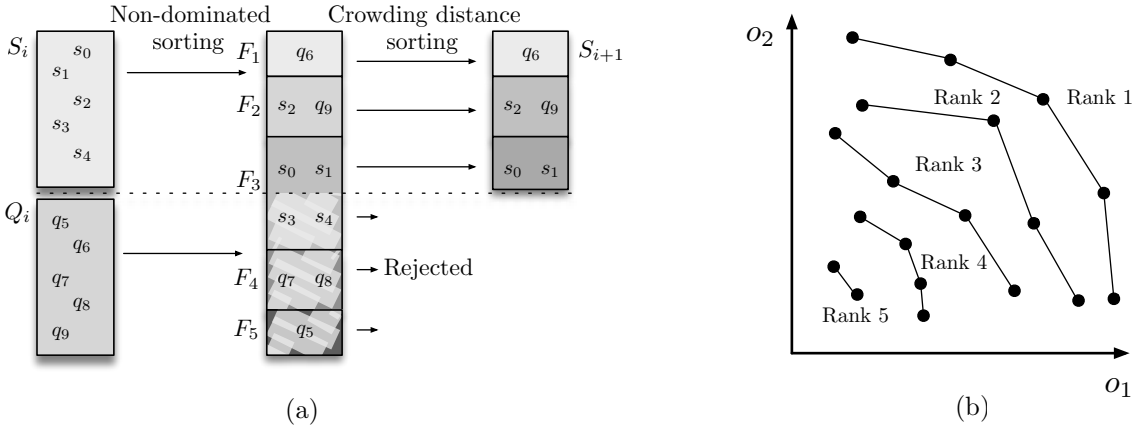


Figure 8.4: (a) Schematic representation of the NSGA-II selection procedure (b) Pareto ranks.

rewards the individuals with unique and less common scores on search objectives. The method also uses an elitist principle, i.e., it selects from the combined population of parents and offspring, rather than from parents only.

The greatest emphasis is however put on the non-dominated solutions. To illustrate this, in Fig. 8.4, we demonstrate how NSGA-II selects candidate solutions at ith generation. In the first step, the population of candidate solutions S_i and the population of their offspring Q_i are combined together to form the selection pool R_i. Next, R_i is partitioned into different Pareto non-dominated fronts (Pareto fronts) which are then used to fill the new population S_{i+1}, one at a time, staring with the first Pareto front F_1. The candidate solutions from the subsequent fronts are then added to S_{i+1} until all available slots in the new population have been accommodated (so that its cardinality is the same as the cardinality of the previous population). Usually, the candidate solutions from the last considered front cannot all fit into S_{i+1}, in which case they are sorted according to the crowding distance in the descending order, and the requested number of top individuals from the ordered list are selected.

NSGA-II uses Pareto dominance to assign rank values to candidate solutions, but there are other possibilities, like for instance domination count [100] and domination strength [369]. There is also a variety of density estimation methods that can be used for diversity promotion. The most commonly used approaches include niching and fitness sharing [100], K-nearest-neighbors [370], and ϵ-domination [167]. In recent years, methods based on fuzzy domination [269] have also been proposed. For more details regarding these and other techniques, we refer the reader to the works of Coello and Deb [56, 73].

8.6 Related concepts

In this section, we discuss several concepts in computational intelligence and machine learning that are to some extent related to the framework of search objectives proposed here.

In EC, the idea of searching for alternative means of characterizing candidate solution's performance is related to *surrogate fitness functions*. Also known as *surrogate models*, *approximate fitness functions*, and *response surfaces* [153], surrogate fitness functions provide a computationally cheap approximation of the original objective function and are thus particularly helpful in domains where, e.g., evaluation involves simulating candidate solutions in some environment/context. While engineered manually in the past, the availability of a multitude of probabilistic modeling and machine learning techniques made the design of surrogate models much easier nowadays, and they are now

commonly built by learning from samples of evaluations. They are particularly popular in continuous optimization, where they can be conveniently implemented using polynomials, Gaussian processes, or artificial neural networks. Occasionally, surrogate models have been also used in GP. For instance, in [131], Hildebrandt and Branke proposed a surrogate fitness for GP applied to job-shop scheduling problems. A metric was defined that reflected the way in which the programs ranked the jobs; when evaluating an individual, the metric was used to locate its closest neighbor in a database of historical candidate solutions, and that neighbor's fitness was used as a surrogate.

Despite conceptual similarities between the notion of search objectives and surrogate fitness functions, there are some key differences that clearly differentiate both concepts. First of all, surrogate-assisted evolutionary search is mainly motivated by the need to reduce computational time in optimization of expensive problems. For this reason, surrogate fitness functions are expected to closely approximate the original objective function. By contrast, the idea of search objectives stems mainly from the observation that an objective function may not provide a suitable gradient for the search, particularly under the MEU solution concept. Secondly, a surrogate fitness function, once integrated with an evolutionary search technique, is typically static and does not change in the course of optimization. The framework of search objectives is on the other hand dynamic, allowing the objectives to change with time and adapt to the current state of search process. Finally, surrogates are typically applied to optimization problems, while the framework for discovery of search objectives, as clearly indicated by the name, focuses on search problems.

One of the highlights of the framework proposed here is its ability to recast a single-objective problem as a multi-objective one. First attempts of performing such a decomposition date back to 1993, when Louis and Rawlins [223] demonstrated that certain deceptive problems could be solved more easily using EAs equipped with a Pareto-based selection. This methodology was later refined by Knowles in [168] and termed *multiobjectivization*. The essence of multiobjectivization is to decompose the problem by introducing additional objectives that convey some problem-specific knowledge. The authors demonstrate that such decomposition tends to eliminate some of the local optima in the fitness landscape and thus makes search algorithms more efficient. Our framework for discovery of search objectives diverges from the concept of multiobjectivization in how these extra objectives are derived and used to guide search. In [168], the objectives are hand-crafted, which usually requires substantial domain knowledge, whereas in our framework search objectives are derived automatically, as we will illustrate in Chapters 9 and 10. Furthermore, the decomposition via multiobjectivization is an one-off process, which implies that the objectives cannot change with time (as in the case of surrogate-assisted evolutionary optimization). With our framework, we put emphasis on changing the objectives dynamically.

As we have already discussed earlier, deriving search objectives transforms a single-objective test-based problem into a multi-objective one. On the other hand, if we consider every test as an 'elementary' objective, our framework can be seen as a method that transforms a *many-objective* problem into a multi-objective one. The recent interest in such transformation techniques [33, 222, 328] is justified by the inferior performance of some multi-objective evolutionary algorithms when more than three objectives are involved [162, 166]. Such *objective reduction* approaches assume the existence of redundant objectives in a given M-objective problem and aim at identifying the smaller (or smallest) set of m ($m \leq M$) conflicting objectives which generates approximately the same Pareto-optimal front as the original problem. They can be categorized into dominance structure-based and correlation-based approaches. In the former case, the redundant objectives are removed [33]; in the latter, the elimination concerns objectives that are non-conflicting, e.g., along the significant eigenvectors of the matrix of correlations between the original objectives [312].

The idea of augmenting a single-objective optimization method with some additional objectives has also surfaced in EC in the form of *helper objectives* [152] that are used along the original objective function (termed primary objective in this context) in a multi-objective setting. The concept of helper objectives is closely related to that of multiobjectivization in employing additional, handcrafted objectives to guide the search. Helper-objectives are typically chosen in a way that helps to avoid local optima, maintains diversity in a population of candidate solutions, and/or supports the recognition of good building blocks (pieces of candidate solutions that positively contribute to its evaluation) that can be exploited by crossover operator. According to Jensen, helper-objectives should ideally remain orthogonal with the primary objective and reflect some aspect of the problem which is expected to prove helpful during the search. For example, in job-shop scheduling, helper objectives could be designed to reflect the flow-times of individual jobs [220]. This sort of consideration requires however deep immersion in problem domain.

The usage of multiple search objectives can be seen as a means for rewarding candidate solutions for being 'original' in their capabilities. This motif has emerged in several works in the past, some of which we already discussed in Section 8.3.1. Another interesting contribution with similar motivations is *novelty search* [208, 254], where a measure of behavioral similarity substitutes for the traditional objective function. The measure is used to reward the individuals who differ significantly from the other individuals in the population and from selected past individuals stored in an archive. Novelty promotes the emergence and coexistence of different skills in a population, and can be seen as an analog to *curiosity* and *intrinsic motivation* in reinforcement learning and developmental robotics [160, 265]. In practice, it must be combined with a 'complexifying' algorithm like NEAT, which takes care of the order in which behaviors are discovered [208], usually stimulating the candidate solution to discover the simple behaviors before the more complex ones.

The idea of dynamically selecting and switching objectives during evolution is closely related to the concept of *incremental evolution* [116]. In this approach, candidate solutions are evolved incrementally by tackling a succession of related *subtasks*. These subtasks implicitly define evaluation functions to be used at a particular stage of evolution. There is therefore a clear relation between subtasks and search objectives, as both provide alternative means to guide a search algorithm. The concept of incremental evolution has been extensively applied in the field of evolutionary robotics [347, 26]. A particularly interesting piece of work in that area is by Harvey [122], who trained robots to distinguish and move towards either triangular and rectangular objects. Noteworthy, the robots learned three subtasks: recognition, pursuit and discrimination between the two geometric shapes.

In coevolutionary algorithms, the closest counterpart of search objectives are coordinate systems (cf. Section 8.3.3). Recall that the dominance relation induced by derived search objectives does not have to be consistent with the original dominance relation defined on G, but instead it is required to create a useful search gradient. Coordinate systems were designed with similar goal in mind, however, they implement an exact approach, i.e., the spatial arrangement of solutions in the CS exactly reproduces the original dominance structure. If that structure was sparse, such was also the structure of the derived CS (which typically manifested in the CS having many dimensions). Furthermore, because the problem of finding the CS of minimal dimensionality is NP-hard [144], and because its dimensionality for even the simplest test-based problems turns out to be very high, using these formalisms to actually drive search is rare and does not lead to significant improvements [143].

Last but not least, there are certain connections between the framework of search objectives proposed here and semantic GP [252] and behavioral GP [191, 187, 183]. In semantic GP, one defines program semantics as the vector of outputs produced by that program for particular tests.

From the viewpoint of our framework, a single row in an interaction matrix is the outcome of confronting program's semantics with the vector of desired outputs. Recent years have seen a large number of contributions that employ this characterization of program behavior to design new initialization, search, and selection operators [251, 350].

8.7 Chapter summary

In this chapter, we introduced the framework for automatic discovery of search objectives, which serves the purpose of broadening the evaluation bottleneck (cf. Section 6.1) and acquiring alternative (or additional) behavioral information from candidate solutions. A fundamental idea behind our framework is that candidate solutions are often complex entities that behave in rich ways, and their behaviors can be exploited to make the search for an ideal solution more effective. We formalize this idea using the concept of interaction matrices that gather outcomes of interactions between candidate solutions and tests, and using such matrices to devise multi-aspect evaluation of candidate solutions.

Chapter 9

Discovery of Search Objectives by Clustering

The framework for discovery of search objectives introduced in the previous chapter addresses the shortcomings of conventional evaluation (cf. Chapter 6) by providing multifaceted characterizations of candidate solutions that can be used as alternative means of driving the search. The core component of any method that subscribes to this framework is a derivation algorithm u that transforms G into a derived interaction matrix G' (Eq. 8.4.6). The columns of G' define a low number of performance measures, which we refer to as search objectives, while their elements are interpreted as scores of candidate solutions on these objectives. So far, we have discussed the proposed framework in isolation, without showcasing any particular derivation algorithm. In this chapter, we propose the first concrete method for discovery of search objectives by heuristic clustering of interaction outcomes (DOC).

This chapter is organized as follows: in Section 9.1, we introduce the method and then, in Sections 9.2 and 9.3, we lay down its formal underpinnings, proving that derived search objectives preserve a great deal of dominance between candidate solutions. Next, in Section 9.4, we present the results of a comparative experiment involving CoEAs and three unrelated test-based domains: Iterated Prisoner's Dilemma [51], Number Games [70], and Density Classification [86]. In Section 9.5, we evaluate DOC's usefulness in GP. We analyze algorithm performance, search dynamics, the number of discovered objectives, and the correlations between them. We conclude the chapter with discussion is Section 9.6 and closing remarks in Section 9.8.

The approach presented in this chapter has been originally published [211] and later extended in [186, 214].

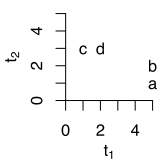
9.1 DOC

In Chapter 6, we showed that achieving certain *combinations* of interaction outcomes can be the key to a more effective search. Based on this observation, we propose Discovery of Search Objectives by Clustering (DOC), a method that identifies the combinations that prevail in the current interaction matrix and uses them to gauge the candidate solutions. This causes novel combinations of test outcomes to be less likely 'forgotten' in the search, even if they are inferior in terms of scalar evaluation.

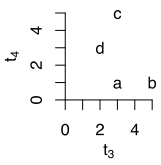
DOC replaces the conventional evaluation stage of an evolutionary algorithm (cf. Section 8.4.3). Given a population S of m candidate solutions and a set T of n tests, it proceed as follows:

G	t_1	t_2	t_3	t_4
a	5	1	3	1
b	5	2	5	1
c	1	3	3	5
d	2	3	2	3

(a)



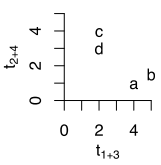
(b)



(c)

G'	t_{1+3}	t_{2+4}
a	4	1
b	5	1.5
c	2	4
d	2	3

(d)



(e)

Figure 9.1: Example of compression of interaction matrix (a) featuring a four-dimensional dominance structure (b, c), into a derived interaction matrix (d), resulting with the dominance structure shown in (e).

1. Calculate the $m \times n$ interaction matrix G between the candidate solutions from S and the tests from T using the interaction function g.
2. Cluster the tests. We treat every column of G, i.e., the vector of interaction outcomes of all solutions from S with a test t, as a point in an m-dimensional space. A clustering algorithm of choice is applied to the n points obtained in this way. The outcome of this step is a partition $\{T_1, \ldots, T_k\}$ of the original n tests in T into k subsets/clusters $T_j \subset T$, $j \in [1, k]$, where $1 \leq k \leq n$ and $\forall_j : T_j \neq \emptyset$.
3. Define the search objectives. For each cluster T_j, we average row-wise the corresponding columns in G. This results in an $m \times k$ derived interaction matrix G', with the elements defined as follows:
$$g'_{ij} = \frac{1}{|T_j|} \sum_{t \in T_j} g(s_i, t) \tag{9.1.1}$$
where s_i is the candidate solution corresponding to the ith row of G.

The columns of G' implicitly define k transient (i.e., adequate only for this specific S and T) search objectives that characterize the candidate solutions in S. The value of jth such objective for a candidate solution s_i is g'_{ij}. Note however that these objectives are derived from the interactions between a specific S and a specific T, and as such are undefined outside S and T. Nevertheless, to emphasize the analogy between the search objectives g' and the original interaction function g, we will alternatively denote g'_{ij} as $g'_j(s_i)$.

The derived search objectives form a multi-objective characterization of the candidate solutions in the context of the current population of tests. They can be subsequently employed by any multi-objective selection method discussed in Section 8.5.

Example 9.1. Consider the matrix of interactions between the population of candidate solutions $S = \{a, b, c, d\}$ and the population of tests $T = \{t_1, t_2, t_3, t_4\}$, shown in Fig. 9.1a. The four-dimensional space of interaction outcomes is shown in two two-dimensional scatter plots (Figs. 9.1b and 9.1c) that span $t_1 \times t_2$ and $t_3 \times t_4$, respectively. The performance of candidate solutions on the tests t_1 and t_3 is quite correlated, and so is t_2 and t_4. Assume the clustering algorithm (step 2 of DOC) notices these similarities and produces $k = 2$ clusters that partition T into $\{T_1, T_2\}$ with $T_1 = \{t_1, t_3\}$ and $T_2 = \{t_2, t_4\}$. Averaging the interaction outcomes within T_js (step 3 of DOC) results in the derived interaction matrix G' shown in Fig. 9.1d. Fig. 9.1e presents the locations of the candidate solutions in the space of derived objectives.

If interaction outcomes are to be maximized, the only dominance holding in the original space is $b \succ a$. In the space of derived objectives (Fig. 9.1e) b still dominates a. However, now also c dominates d, thought originally these two solutions were mutually non-dominated (incomparable). As a result of 'compressing' of the original interaction matrix, some information about the dominance structure has been lost.

In the particular case of c and d, introducing dominance in favor of c may be desirable, as c outperforms d on two original tests (t_3, t_4), while only one test (t_1) supports the opposite relation (and t_2 is neutral in this respect). DOC trades the lower number of resulting objectives for certain inconsistency with the original interaction outcomes. Nevertheless, we posit that this imprecision may be a price worth paying for obtaining a potentially useful search gradient. In Section 9.3, we present a detailed analysis of dominance preservation. ■

DOC broadens the bottleneck of evaluation in characterizing the candidate solutions with k objectives rather than with a single one (cf. motivations in Chapter 6). On the other hand, using small k is likely to cause the dominance relation in the derived space to be dense enough to provide a reasonably strong search gradient (as opposed to the dominance on all tests, discussed next to Eq. 6.2.1). As a result, candidate solutions that feature different 'skills' (embodied by particular search objectives) can coexist in a population, even if some of them are clearly better than others in terms of scalar evaluation. For instance, solution c in the above example is not dominated in G', although its scalar fitness (12) is lower than that of the most fit solution b (13).

Because DOC drives selection using multiple search objectives, it might seem appropriate to pair it not with the solution concept of MEU, but with that of Pareto optimality (cf. Section 5.3). We claim however that this convergence is only apparent. Notice first that the desired objectives are transient, derived in every generation independently, from a usually different interaction matrix. Therefore, one cannot claim that DOC optimizes for any specific objectives throughout an entire run. Moreover, the solution concept of Pareto optimality comes in handy when one aims at finding a *set* of candidate solutions that possibly closely approximate the non-dominated ones and exploit the trade-off between the objectives. This is not the case in the class of problems we consider in this thesis: our goal is to find *one* solution that maximizes the odds of passing *any test*, for which MEU is the most appropriate solution concept.

9.2 Properties of DOC

In Section 8.4.3, we discussed several properties that characterize search objectives and methods for their discovery. DOC naturally conforms to several of these properties. In particular, evaluation conducted by DOC is contextual: as in all algorithms solving test-based problems, the outcome of evaluation of a candidate solution in S depends on the current tests in T. However, that outcome depends also on the other candidate solutions in S, because they together determine the result of clustering. Notice that such an interaction between candidate solutions is not common in EAs.

As the clustering algorithm partitions the set of tests T, rather than, e.g., selecting some tests, none of the tests are discarded, and each of them influences one of the derived objectives. Also, clustering guarantees that the tests that are mutually redundant (i.e., identical columns in G) will support the same search objective (cf. Section 8.4.5). In general, the more tests are similar in terms of solutions' performance on them, the more likely they will end up in the same cluster and contribute to the same search objective. Clustering also guarantees insensitivity to permutations of rows and columns in G, and basic algebraic operations, i.e., scaling and translation (though the latter depends on the internals of the clustering algorithm and the metric used therein).

For $k = 1$, DOC degenerates to a single-objective approach: all tests form one cluster, and G' has a single column that contains solutions' (normalized) estimate of expected utility defined in (5.3.1). Setting $k = n$ implies $G' = G$, and every derived objective being associated with a single test.

Table 9.1: Dominance relation $\succ_G$ in the test space and dominance relation $\succ_{G'}$ in the space of derived objectives. Note that the non-dominance case in the derived space ($s_i \not\succ_{G'} s_j \wedge s_j \not\succ_{G'} s_i$) includes also indiscernibility, i.e., the case $\forall r : g'_r(s_i) = g'_r(s_j)$.

	$s_i \succ_{G'} s_j$	$s_i \not\succ_{G'} s_j \wedge s_j \not\succ_{G'} s_i$	$s_j \succ_{G'} s_i$
$s_i \succ_G s_j$	Preserved	FN	Inversion
$s_i \not\succ_G s_j \wedge s_j \not\succ_G s_i$	FP	Preserved	FP
$s_j \succ_G s_i$	Inversion	FN	Preserved

The derived objectives sum up to the scalar fitness (5.3.2), i.e., $\sum_{j=1}^{k} g'_{ij} = f(s_i)$, and each of them is a linear combination of selected columns in G. This feature is essential for dominance preservation, as we demonstrate in the next subsection.

9.3 Preservation of dominance

As demonstrated in Example 9.1, the objectives derived by DOC are lossy, i.e., the dominance relation in the space of derived objectives is in general different from the dominance relation in the space of original objectives. In this section, we investigate the nature of those differences.

Consider a pair of candidate solutions s_i,s_j with different interaction outcomes, i.e. $\exists t \in T : g(s_i, t) \neq g(s_j, t)$. Table 9.1 lists the combinations of the possible relations between s_i and s_j in the original space of interaction outcomes ($\succ_G$) and in the space of the derived objectives ($\succ_{G'}$). The cases in which the relation is consistent in both spaces are marked by 'Preserved'. In the remaining combinations, three types of errors may take place, two of which can be likened to errors in statistical tests:

- False positive errors (FP), when one of the solutions dominates the other in G', although they were incomparable (mutually non-dominated) in G,
- False negative errors (FN), when a dominance in G ceases to hold in G',
- Inversions of dominance, when one of the solutions dominates the other in G, while G' supports the opposite relation.

Of these categories, inversions are clearly the most severe mistakes, as they may deceive selection and so misguide the search process. In the following, we show that DOC cannot commit this kind of errors.

Theorem 9.2. *Let s_i, s_j be candidate solutions such that $s_i \succ_G s_j$, $i, j \in \{1, \ldots, m\}$. Then $s_i \succ_{G'} s_j$.*

Proof. . Let us assume that $s_j \succ_{G'} s_i$. Thus for some $r = 1, \ldots, k$ there holds $g'_r(s_i) < g'_r(s_j)$, then by (9.1.1) we have that

$$\frac{1}{|T_r|} \sum_{t \in T_r} g(s_i, t) < \frac{1}{|T_r|} \sum_{t \in T_r} g(s_j, t),$$

where $T_r \in \{T_1, \ldots, T_k\}$. The above inequality is equivalent to

$$\sum_{t \in T_r} (g(s_i, t) - g(s_j, t)) < 0.$$

For this to hold, at least one of the summed terms has to be negative. This is however impossible, as $s_i \succ_G s_j$, i.e $g(s_i, t) \geq g(s_j, t)$ for all $t \in T$. Therefore $s_i \succ_{G'} s_j$. □

DOC will never cause dominance inversion, and is in this sense non-deceptive.

Theorem 9.2 implies that DOC cannot commit false negative errors. For $s_i \not\succ_{G'} s_j \wedge s_i \not\succ_{G'} s_j$ to hold in G', at least one derived objective g'_r has to reverse the ordering of the solutions in question (i.e., $g'_r(s_i) < g'_r(s_j)$), which, as we have shown above, is impossible.

To investigate the false positive errors, we assume $s_i \not\succ_G s_j \wedge s_j \not\succ_G s_i$. The absence of dominance implies that there exists at least one test t such that $g(s_i,t) < g(s_j,t)$ and at least one test t' such that $g(s_i,t') > g(s_j,t')$. Let us assume that these are the only tests in T and that a single derived objective g'_1 is built from them, i.e.,

$$g'_1(s) = \frac{1}{2}(g(s,t) + g(s,t')).$$

The presence and direction of dominance between s_i and s_j in the derived space will depend on the sign of $g'_1(s_i) - g'_1(s_j) = \frac{1}{2}(g(s_i,t) - g(s_j,t) + g(s_i,t') - g(s_j,t'))$. Clearly, depending on how much s_i is worse than s_j on t and better than s_j on t', the sign of that expression can be arbitrary. This holds also if there are other tests in T.

Therefore, DOC can commit false positive errors, i.e., posit a dominance in G' for a pair of solutions when it is factually absent in G. This error is unavoidable, given the reduced dimensionality of G'[1]. On the face of it, this may be considered undesirable. However, note that, for most problems T is only a sample from a universe of tests $\mathcal{T}$, so even G does not fully characterize the candidate solutions (cf. Section 5). As discussed in Section 8.4.4, attempting to perfectly preserve the dominance in G' may be not worth the effort, given that G captures only a partial characteristics of the solutions in S. By the same token, we do not find it critical that the clustering algorithms employed by DOC are heuristic, and thus may produce sub-optimal groupings of tests.

9.4 Experimental analysis in the domain of CoEAs

The following computational experiment is intended to quantify the impact of search objectives derived by DOC on the efficiency of coevolutionary learning in test-based problems representing various domains. Our secondary objective is to inspect the resulting objectives and cast some light on the dynamics of search process driven by them.

9.4.1 Basic coevolutionary configurations

All configurations described in the following implement the generational two-population coevolutionary algorithm, with separate populations of candidate solutions S and tests T. In every generation, each candidate solution $s \in S$ interacts with every test $t \in T$, producing an interaction matrix G (cf. Section 5). The configurations differ only in the way the fitness values of the individuals in S are calculated from G.

In the conventional coevolutionary learning (CEL), our reference method, the fitness of a candidate solution s is the estimate of expected utility (Eq. 5.3.2) calculated from the interactions with all tests in T, i.e., the average of the corresponding row of G. This fitness is then used by a conventional scalar selection operator (tournament of size 5 or or $\mu+\lambda$, depending on the domain; Section 9.4.4).

DOC (Sect. 9.1) clusters the columns of the interaction matrix G and produces $k \geq 1$ objectives g_j that characterize the candidate solutions in S. To avoid fixing k in advance, we decided to employ X-MEANS [280], an extension of the popular k-means algorithm. Given the admissible range of

[1] In general, the likelihood of committing such errors grows with the ratio of the number of tests (dimensionality of the original dominance space) to the number of derived objectives.

k, X-MEANS picks k that produces the clustering that maximizes the Bayesian Information Criterion. In this experiment, we allow X-MEANS consider $k \in [1, 5]$ and employ the Euclidean metric to measure the distances between the observations (the columns of G).

The resulting k objectives necessitate a multiobjective selection mechanism (cf. Section 8.5.4). Therefore, DOC employs NSGA-II, the arguably most popular multi-objective selection algorithm [75]. We use NSGA-II rather than recently proposed NSGA-III [74] due to questionable performance gains offered by the latter [140].

Concerning the evaluation of tests, CEL, DOC and the other configurations presented in the following rely on distinctions (cf. Section 5.5.1). The fitness $\hat{Q}_S(t)$ of a test $t \in T$ is the number of pairs of candidate solutions in S it differentiates:

$$\hat{Q}_S(t) = |\{(s_1, s_2) \in S \times S : g(s_1, t) \neq g(s_2, t)\}| \tag{9.4.1}$$

$\hat{Q}_S$ promotes the tests that *inform* the candidate solutions about the differences between them, rather than how they *perform*. Distinctions proved effective at maintaining the *coevolutionary gradient* in many nontrivial test-based problems [70, 337, 27].

9.4.2 Additional control configurations

Comparing DOC to the CEL cannot fully explain the anticipated differences, because there are *two* aspects that differentiate these methods. Firstly, NSGA-II employed by DOC is more sophisticated than the scalar selection used in CEL: not only it operates on a combined pool of parents and offspring (and is thus quite elitist), but essentially performs a two-stage selection by first rejecting the lower ranks of the Pareto-ranking, and then employing tournaments on ranks and crowding to draw the parents (cf. Section 8.5.4). Secondly, the sole fact of having *any* multiple objectives causes selection to operate differently, because the likelihood of dominance between solutions decreases with the number of objectives, and many inconclusive comparisons result in weaker selection pressure.

To separate these aspects, we design two additional control setups. To control for NSGA-II selection, we design the 1-MEANS setup, which is simply DOC with the numbers of clusters $k = 1$. Technically, there is no need to run clustering in this setup as all tests by definition belong to the same cluster. In this configuration, selection of candidate solutions is driven by a single derived objective g_1 which, by averaging the outcomes on all tests in T, is equivalent to the expected utility (Eq. 5.3.1; cf. Section 9.2), i.e., the fitness used by CEL. However, contrary to CEL that relies on scalar selection, 1-MEANS employs NSGA-II, and thus still involves ranking.

To control for the presence of multiple objectives, we provide the RAND configuration, which replaces DOC's clustering of the interaction matrix G with the following steps. First, RAND draws k from $[1, 5]$ at random. Then it randomly partitions the columns of G into k objectives, which are then treated in the same way as in DOC. Thus, contrary to 1-MEANS, RAND relies on multiple objectives (unless the drawn k happens to be 1), but those objectives in general do not reflect any similarities between the columns of G. This configuration should help us verify if discovering *meaningful* clusters is essential[2].

9.4.3 Extensions of DOC

The basic DOC algorithm clusters the interaction outcomes using the Euclidean distance which allows handling continuous as well as multi-valued interaction outcomes (for instance, the IPD

[2]Note that the objectives derived by RAND, by being selected at random, are estimates of expected utility in the same sense as $\hat{Q}_T$ (Eq. 5.3.2), albeit based on smaller samples than the entire population of tests T. Also, similarly to DOC, they sum up to scalar fitness.

benchmark presented below features ternary interaction outcomes). However, using the Euclidean distance may render it difficult to capture the *combinations* of skills exhibited by particular solutions.

Consider a population hosting two candidate solutions $S = \{s_1, s_2\}$. In such a case, clustering takes place in a two-dimensional space, with dimensions corresponding to s_1 and s_2. Assume that clustering leads to two clusters with the centroids in $m_1 = [0.8, 0.9]$ and $m_2 = [0.5, 0.4]$. Now consider a test t with the interaction outcomes $[0, 1]$, i.e., $g(s_1, t) = 0$ and $g(s_2, t) = 1$. The Euclidean distance between t and m_1 amounts to $d(t, m_1) = \sum_i (t(i) - m_1(i))^2 \approx 0.65$ ($i \in \{1, 2\}$ iterates over dimensions; we omit the square root for simplicity), while $d(t, m_2) \approx 0.61$. Therefore, t will be assigned to the second cluster. However, that cluster groups tests that are more often solved by s_1 than by s_2 ($0.5 > 0.4$). One may argue that the first cluster is more suitable for t, as it hosts the tests with complementary characteristics ($0.8 < 0.9$).

To address this problem, we consider two extensions of DOC. In DOC-BIN, we binarize the centroids, i.e., the distance is defined as $d'(t, m) = \sum_i (t(i) - [m(i)])^2$, where $[m(i)]$ denotes rounding $m(i)$ the closest integer. For the example above, $d'(t, m_1) = 1$ and $d'(t, m_2) = 2$, so t would be assigned to the first cluster.[3]

By rounding the centroids' coordinates to the closest integer, DOC-BIN implicitly assumes that the tests in a cluster should be characterized by 0 on the ith dimension if less than half of the tests in the cluster are solved by s_i, and by 1 otherwise.

However, the above reasoning ignores the fact that a strong solution will solve *most* (and not half) of the tests in T (and, conversely, a weak solution will solve hardly any tests). Setting the threshold to 50 percent seems thus rather arbitrary; a more adequate value can be estimated from the performance of s_i on all tests. Therefore, in the last configuration, DOC-AVG, the distances are calculated according to formula $d''(t, m) = \sum_i (t(i) - [m(i)]_g)^2$, where

$$[m(i)]_g = \begin{cases} 0 & if\ m(i) < \frac{1}{|T|} \sum_{t \in T} g(s_i, t) \\ 1 & otherwise \end{cases}.$$

9.4.4 Test problems

The suite of benchmarks consists of Iterated Prisoner's Dilemma (IPD, Section 5.4.3) [13], Numbers Games (NGs, Section 5.4.2) [353] and Density Classification Task (DCT, Section 5.4.4) [66], elegant and well-defined problems that have been widely used to analyze evolutionary learning algorithms. In particular, they are excellent testbeds for coevolutionary algorithms due to their test-based nature and large (and for NGs infinite) number of tests. Popularity of these benchmarks for competitive coevolution stems primarily from their high difficulty, manifested in the failure to obtain quality solutions with generic metaheuristics.

Another particularly appealing feature is their kinship to complex real-world scenarios. For instance, IPD is widely used to model systems in biology [259], psychology [305], and economics [130]. Recently, it gained even more popularity as a demanding task for competitive environments [245, 64, 51, 53]. Cellular automata used in DCT have applications in many fields including CPUs, cryptography [361], and real-world biological and chemical systems [362, 163]. DCT is also very difficult, as witnessed by the relatively slow improvement of best-known solutions over time [286, 86, 365]. Finally, NGs were designed to objectively measure the performance of coevolu-

[3] For binary domains, DOC-BIN with the Euclidean distance is equivalent to DOC with the Hamming distance (assuming centroids were determined by the mode in place of the arithmetic mean). However, this is not true for multi-valued interaction functions like in the IPD benchmark.

tionary algorithms and determine whether they are vulnerable to coevolutionary pathologies (cf. Section 3.7).

As IPD and NGs are symmetric games, we employ the same strategy representation for candidate solutions and tests. In the asymmetric DCT, different representations are necessary. The detailed description of each benchmark in the experimental suite can be found in Section 5.4. In the following, we only provide the details regarding the experimental setup.

Iterated Prisoner's Dilemma

We use IPD with $c = 9$ choices, which we found to be much more demanding than the 3-choice IPD used in [53]. Each strategy is a look-up table of $9 \times 9 + 1 = 82$ moves and the size of search space is $9^{82} \approx 1.77 \times 10^{78}$. We make each IPD game consist of 150 PD episodes.

For IPD, all configurations maintain populations of 50 candidate solutions and 50 tests. However, because NSGA-II effectively merges the parents and the offspring prior to selection, we set the size of candidate solution population to 100 for CEL. This provides for fair comparison: every method engages $100 \times 50 = 5{,}000$ IPD games per generation. With runs lasting for 200 generations, the total effort per run amounts to 1,000,000 games.

Both populations are initialized with uniformly randomized look-up tables. For selection in the single-objective methods, tournament of size 5 is used. The only source of genetic variation is a mutation operator, which iterates over all elements of a look-up table and with probability 0.2 replaces the original choice (move) with one of the remaining $c - 1$ choices selected at random. This operator has been found to provide sufficient variation for multiple-choice IPD [51].

Numbers Games

We consider $l \in \{3, 4, 5\}$-dimensional variants of COA and COO (cf. Section 5.4.2). Following [70], for both populations, the initial values in each dimension are uniformly sampled from $[0, 0.1]$. Offspring individuals are created from the parents by picking at random two dimensions from $[1, l]$, and adding to them a random value x chosen uniformly from $[-0.1, 0.1]$. Both populations host 200 individuals each. For selection, we use the $(\mu + \lambda)$ evolution strategy [22], with $\mu = 100$ and $\lambda = 100$. Each evolutionary run lasts for 1000 generations.

Density Classification Task

The initial population of CA rules contains lookup tables drawn at random. As drawing the ICs in uniformly causes them to be usually very difficult (the expected number of ones is close to $\frac{l}{2}$), they require a more sophisticated initialization. First, a number d is uniformly sampled from the interval $[0, l]$. Then, a vector of length l is filled with 1s on the first d positions and 0s on the remaining positions. Finally, the vector is randomly shuffled. As a result, the number of ones per IC is uniformly distributed in the population of tests.

Rules and ICs are varied by a 2% and 5% per-bit mutation rate, respectively. Both populations (of candidate solutions and tests) host 200 individuals each. For selection, the $(\mu + \lambda)$ evolution strategy [22] is used. An evolutionary run lasts for 200 generations, resulting in the total effort of 8,000,000 interactions. We consider three DCT instances of various difficulty: an easy one (DCT-1, $l = 31$, $r = 2$, $p = 110$), studied in [260], a medium one (DCT-2, $l = 59$, $r = 3$, $p = 110$), used among others in [**?**], and a hard one (DCT-3, $l = 149$, $r = 3$, $p = 320$), investigated in [157].

Table 9.2: Average objective performance (estimated expected utility) of the best-of-run individuals, averaged over 60 evolutionary runs. Bold font marks the best result for each benchmark.

	CEL	1-MEANS	RAND	DOC	DOC-BIN	DOC-AVG
IPD	78.30 ± 1.33	83.54 ± 1.72	88.98 ± 1.19	90.76 ± 1.12	91.15 ± 1.19	91.43 ± 0.99
DCT-1	59.77 ± 3.38	79.02 ± 2.69	81.10 ± 2.17	89.53 ± 0.67	83.37 ± 2.49	89.33 ± 1.03
DCT-2	53.02 ± 2.30	56.56 ± 3.51	60.12 ± 3.91	71.33 ± 3.51	64.89 ± 4.83	77.24 ± 3.66
DCT-3	50.18 ± 0.05	50.21 ± 0.05	50.12 ± 0.05	51.45 ± 1.83	49.51 ± 2.08	53.26 ± 2.81
COO-3	86.61 ± 3.64	53.02 ± 4.73	49.73 ± 4.64	89.59 ± 4.32	96.27 ± 2.89	94.76 ± 2.87
COO-4	50.97 ± 1.30	29.19 ± 2.39	26.72 ± 1.60	54.70 ± 3.85	59.63 ± 4.26	53.98 ± 3.94
COO-5	36.46 ± 1.79	20.61 ± 0.92	22.47 ± 1.66	37.48 ± 2.00	37.16 ± 2.35	35.39 ± 2.39
COA-3	83.42 ± 6.31	58.08 ± 8.10	51.34 ± 7.93	93.00 ± 4.92	96.58 ± 3.45	89.08 ± 6.17
COA-4	23.30 ± 4.59	22.32 ± 5.69	20.51 ± 5.37	30.56 ± 6.89	39.36 ± 7.61	35.63 ± 6.67
COA-5	3.97 ± 0.87	6.0 ± 1.88	8.24 ± 2.39	8.77 ± 2.67	12.06 ± 4.05	18.14 ± 5.3

9.4.5 Performance

The assessment of candidate solutions in S, whether single-objective in CEL or multi-objective in DOC, depends on the current state of the population of tests T and is thus *subjective*. As a result, a candidate solution's fitness may strongly differ from its true performance. The *objective* performance measure for all test problems considered here is the expected utility (Eq. 5.3.1). To estimate it, we let a candidate solution interact with 50,000 random tests, generated by the domain-specific procedures used for initializing the population of tests (see the previous three sections). We assess so the best-of-run individuals, i.e., the candidate solution in the last population with the highest subjective fitness. This measurement does not affect algorithms' behavior.

Table 9.2 presents the expected utility of the best-of-run solutions for particular benchmarks and methods averaged over 60 evolutionary runs, accompanied by 95% confidence intervals. To compare the methods on all benchmarks simultaneously, we employ the Friedman's test for multiple achievements of multiple subjects [159]. Compared to ANOVA, it does not require the distributions of variables in question to be normal.

Friedman's test operates on average ranks, which for the considered methods are as follows:

DOC-AVG	DOC-BIN	DOC	CEL	1-MEANS	RAND
1.9	2.2	2.3	**4.7**	**4.9**	**5.0**

The p-value is $\ll 0.001$, which indicates that at least one method performs significantly different from the remaining ones. The bold font marks the methods that are outranked at 0.05 significance level by *all* DOC *variants* (the first *three* methods in the ranking) according to post-hoc analysis using symmetry test [135].

The derived objectives allow DOC to outperform the standard coevolutionary search (CEL) and the other two control setups (1-MEANS, RAND) not only on aggregated ranks, but also on every benchmark, regardless of problem difficulty. This result is often further improved by DOC-BIN and DOC-AVG. For the conceptually more challenging benchmarks, i.e., IPD and DCT, DOC-AVG fares consistently the best. For the abstract number games, the ranking of DOC variants is less predictable.

Interestingly, for some benchmarks we observe also positive influence of decomposing the scalar fitness function by random clustering (RAND). Nevertheless, the inferior overall performance of RAND suggests that the objectives that result from random partitioning of interaction matrix do not provide as efficient search gradient. In Numbers Games, such a blind extraction of objec-

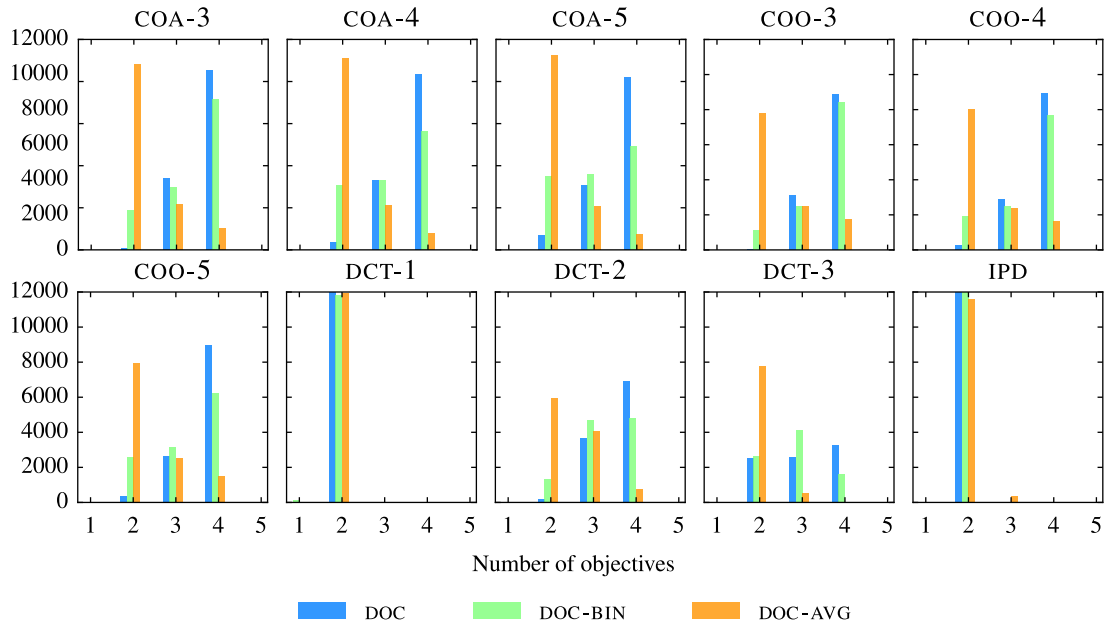


Figure 9.2: The histogram of search objectives in DOC for every benchmark problem. RAND is not included as it features uniform distribution of k.

tives even misleads the search algorithm and ultimately leads to worse performance than simple coevolution (CEL).

The other control setup, 1-MEANS, aimed at isolating the impact of NSGA-II selection, achieves the worst performance in almost every benchmark. Thus, while NSGA-II selection is the crucial part of DOC, using it in a single-objective setting does not translate into an improved search performance.

The obtained results support our claim than DOC is capable of meaningful grouping of tests *and* exploiting the resulting derived objectives in a multi-objective setting. The superiority of all DOC variants with respect to all three control setups (CEL, 1-MEANS, RAND) corroborates our hypothesis that better performance can be achieved only by simultaneous involvement of these two capabilities.

9.4.6 Number of derived objectives

In DOC, the X-MEANS clustering algorithm dynamically adjusts the number of clusters (and search objectives) k to the actual interaction matrix. This number may convey certain information about problem characteristics. In Figure 9.2, we present the histograms of k for every benchmark, gathered from 200 generations of all 60 runs of all three DOC variants. Given that the DOC configurations overall outperformed the control ones, the observed values of k should be considered as having positive impact on method performance. The graphs reveal that for easier problems such as IPD and DCT-1, a lower number of objectives is sufficient to effectively improve the search performance, while the harder ones typically benefit from greater values of k.

Figure 9.2 shows that X-MEANS rarely sets $k = 1$; in fact, we observe this happening only for COA. Apparently, interaction outcomes can be usually better captured using more than one cluster (at least in terms of the Bayesian Information Criterion used by X-MEANS). Also, as this is accompanied by high performance of DOC, we may say that operating in a multidimensional

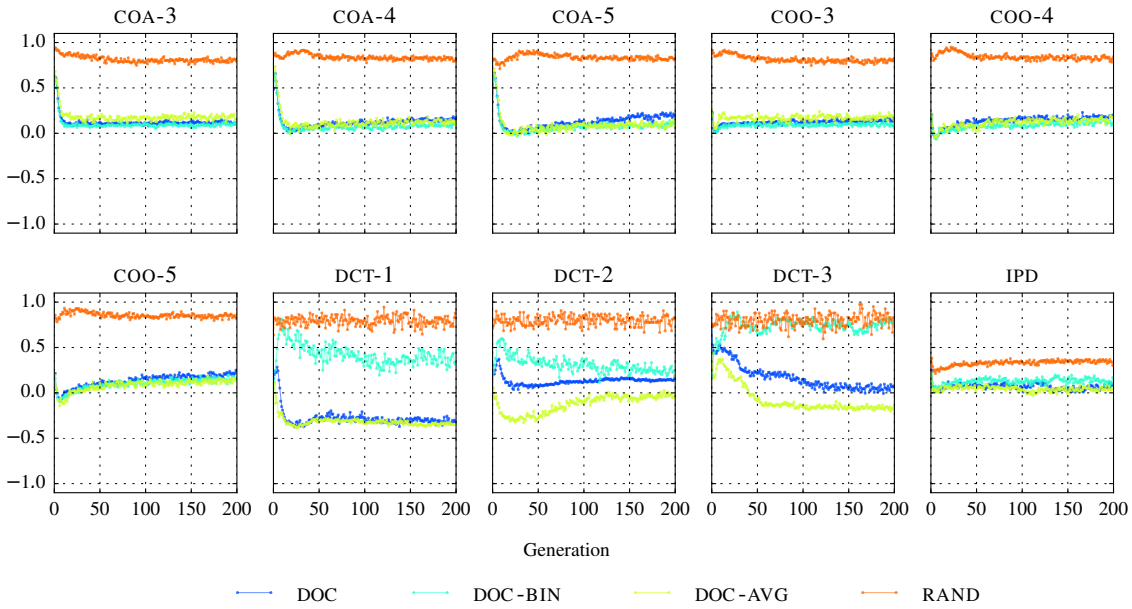


Figure 9.3: Pearson's correlation between the discovered search objectives averaged generation-wise over all runs and all pairs of search objectives.

objective space is in a sense favored by the approach. On the other extreme, $k = 5$ objectives are never derived, suggesting that this upper limit was a good choice for the problems studied here.

Finally, it may be interesting to note that both DOC and DOC-BIN have the tendency to maintain a greater number of extracted objectives, while DOC-AVG has the opposite property. We should however admit that these considerations have to be taken with a grain of salt, as the particular ks observed in Figure 9.2 result not only from the coevolutionary dynamics, but also from the particular measure (Bayesian Information Criterion) used by X-MEANS.

9.4.7 Correlation of objectives

As argued in Chapter 6, we anticipate that nontrivial problems feature mutually-exclusive underlying objectives ('skills'), i.e., such that it is difficult to simultaneously make progress on all of them. It is thus interesting to ask whether such 'polarization' becomes reflected in the search objectives derived by DOC.

To quantify the dissimilarity between any two objectives g'_1 and g'_2 discovered by DOC for a given population of candidate solutions S and a population of tests T, we employ the Pearson linear correlation coefficient calculated for the candidate solutions in S, i.e., the correlation between the two corresponding columns in the compressed matrix G' of interactions between S and T. In Fig. 9.3, we present the correlation calculated in this way, averaged over all pairs of objectives (columns of G') in a given generation of an evolutionary process and across all 60 evolutionary runs. The correlation of the objectives discovered by DOC is usually much lower than the correlation for RAND. Because RAND partitions T randomly, each objective it defines is based on a random sample of T, and the averages calculated from such samples tend to be similar and thus exhibit high correlation. DOC, on the other hand, attempts to find a partitioning of G that minimizes the within-cluster variation, i.e., the amount by which the columns of G within a cluster differ from each other. The objectives it discovers are thus likely to be significantly different from each other and capture diversified aspects of solution's capabilities. This is particularly evident for DCT and COA, where the correlation of objectives discovered by DOC gradually decreases in the early stage

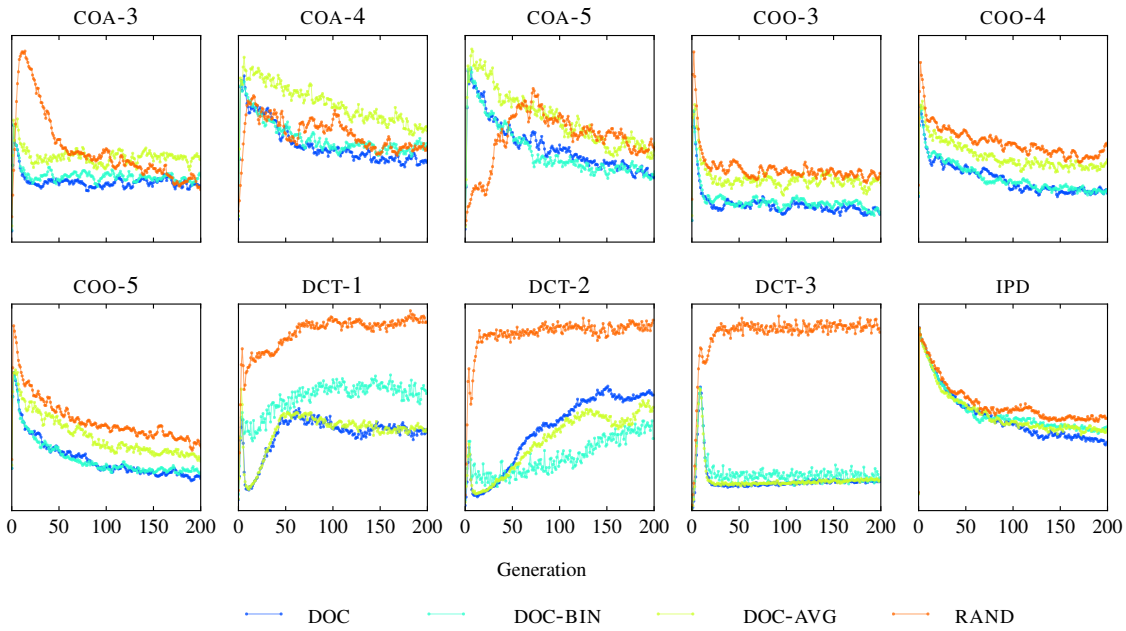


Figure 9.4: Average within-cluster sum of squares (WSS) between the discovered search objectives.

of evolution and then stabilizes. Interestingly, for DCT, the correlation becomes even negative, in which case an improvement on one objective causes a deterioration on the other. These observations corroborate the hypothesis about the mutually exclusive character of the derived search objectives.

9.4.8 Intra- and inter-cluster variance

Correlation characterizes the relationship *between* the search objectives, without actually considering how 'well defined' they are internally. To investigate this aspect, we inspect the derived objectives using the tools characteristic for cluster analysis: within-cluster variance and between-cluster variance of the clusters associated with search objectives. We define the within-cluster sum of squares (WSS) as:

$$WSS = \sum_i \sum_{x \in T_i} ||x - m_i||^2,$$

where T_i is i-th cluster and m_i is its centroid (calculated using arithmetic mean). Lower WSS implies greater similarity between the observations in clusters.

In Fig. 9.4, we present the WSS averaged across all 60 evolutionary runs, plotted against the generation number. The clusters discovered by DOC are typically more compact than those resulting from the random partitioning performed by RAND. This is not surprising, since random grouping is destined for a larger variance. What is more interesting though is the prevailing decreasing trend over the course of evolution, which suggests that the objectives gradually converge towards specific skills revealed by the candidate solutions. WSS decreases also for RAND; however, Table 9.2 showed that this trend is not accompanied with a good performance. This indicates that the objectives discovered by DOC are indeed meaningful, i.e., capable of creating a useful search gradient for the candidate solutions.

The decrease of WSS is not a rule however: we observe just the opposite trend for, e.g., DCT. We hypothesize that this may be attributed to candidate solutions initially exhibiting very similar behaviors (when, for instance, the tests in the first generations turn out to be too difficult for most candidate solutions in the population).

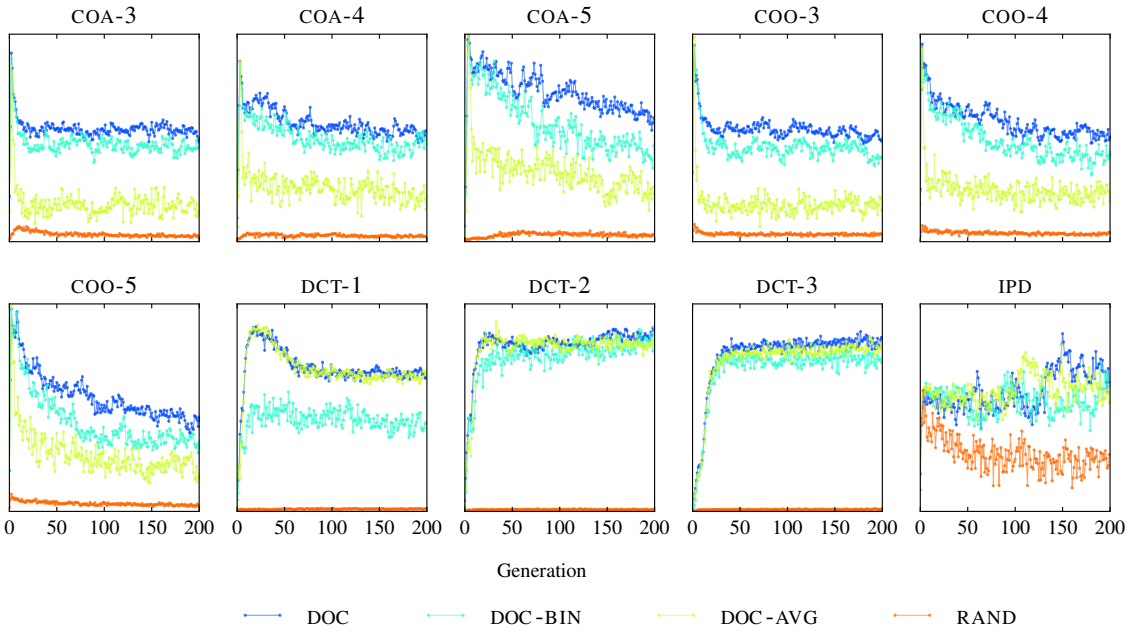


Figure 9.5: Average between-cluster sum of squares (BSS) between the discovered search objectives.

The between-cluster sum of squares (BSS) measures how distinct and well-separated the clusters are from each other:

$$BSS = \sum_i n_i ||m - m_i||^2$$

where n_i is the size of the i-th cluster and m is the global mean of the data. Fig. 9.5 presents the BSS of the derived objectives averaged across 60 evolutionary runs. BSS for RAND is close to zero most of the time. As for WSS, this was expected, given the probabilistic nature of the partitioning performed by RAND. For all variants of DOC, we observe relative stabilization of BSS, typically preceded by a gradual decrease. However, for DCT, we observe a rapid increase of BSS in the early stages of evolution. For DCT-1, the easiest instance of the problem, this is followed by a slight drop. In both domains, pure DOC achieves the highest BSS, though DOC-AVG and DOC-BIN are not far behind. Interestingly, in case of NGs, the situation is quite different: BSS is initially high, suggesting that the objectives are very diverse. Over time, it gradually decreases, causing the objectives to lose their distinct character. The decrease of BSS co-occurs also with the slight rise of the correlation between search objectives (cf. Fig 9.3).

9.4.9 Visualization of the search objectives

Correlations and within- and between-cluster variance provide only cursory information about the derived objectives. A deeper insight can be provided by presenting them graphically. Figures 9.6 and 9.7 visualize the objectives derived by DOC in the last generation of a selected single run for every benchmark. The following procedure was used to create the graphs. First, we scanned the final populations of evolutionary runs in search of compressed interaction matrices G' that had $k = 2$ columns. For each row in G', a green point marks the performance of a candidate solution on the two search objectives. The labels on the axes indicate the number of tests that contributed to the corresponding objective (i.e., the sizes of the corresponding clusters).

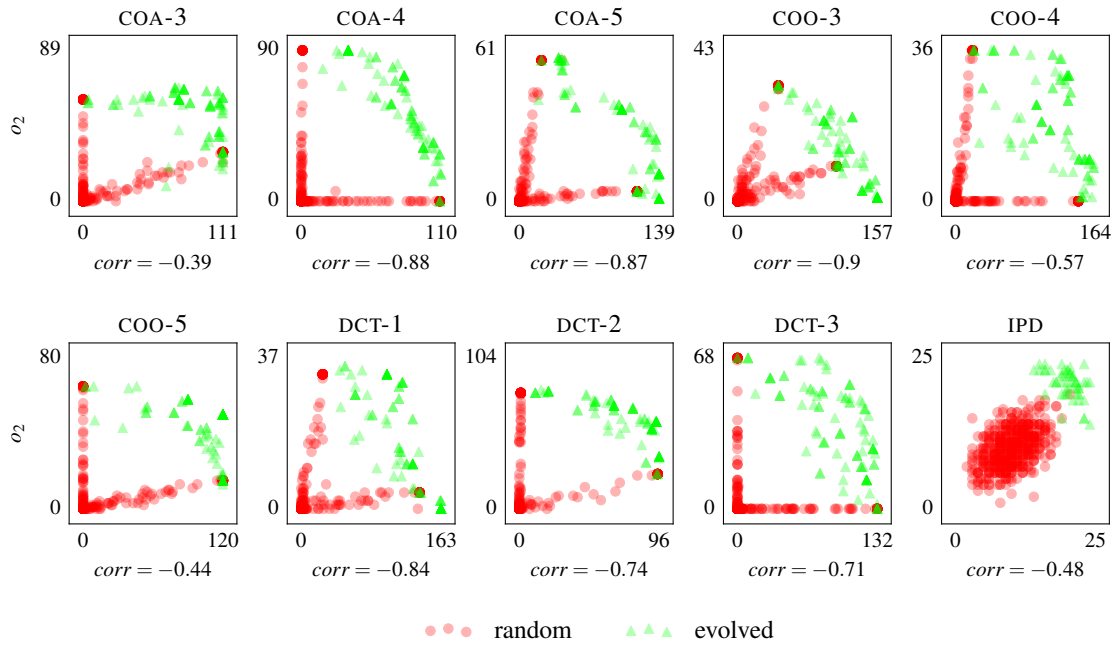


Figure 9.6: Visualization of the negatively correlated search objectives. The evolved candidate solutions marked in green, the random ones in red.

In Fig. 9.6, we group the graphs for the runs that ended with negatively correlated derived objectives. To this aim, we use the Pearson correlation coefficient of the performance of candidate solutions on the search objectives. Due to the coevolutionary nature of DOC, the final candidate solutions in S are adapted to the tests in the final T, so the green marks in Fig. 9.6 reflect only certain combinations of performances on the search objectives. To illustrate the characteristics of the search objectives in a more unbiased way, we plot also the performance of random candidate solutions. To this aim, 5000 random candidate solutions are generated using the problem-specific procedures described in Section 9.4.4. The performance of each such solution on the search objectives is measured by performing interactions with the tests that gave rise to the two objectives in G', and averaging the outcomes within the two clusters (see Eq. 9.1.1). The points obtained in this way are then plotted in red. Where the marks overlap, color saturation reflects their density.

Each inset in Fig. 9.6 corresponds to a different pair of search objectives, specific to a problem being solved, a run, and the state of both coevolving populations at the end of run. The common feature of all graphs is that the evolved solutions stretch between the axes of objectives, each of them differently exploiting the trade-off between the objectives. In doing so, they clearly attempt to approximate the Pareto-front. DOC is clearly able to maintain diversity in a population till the very end of evolutionary runs and so mitigates premature convergence.

The random solutions, on the other hand, typically perform well only on one objective each. Furthermore, they are usually dominated by the evolved candidate solutions and only some of them come close to the Pareto fronts of the evolved candidate solutions. The number of random individuals that manage to achieve a non-zero performance on both derived objectives is relatively small.

The spatial arrangements of solutions shown in Fig. 9.6 are characteristic for runs which ended with decorrelated objectives and as such meet our expectations about the behavior of the method. Nevertheless, occasionally DOC derives uncorrelated or positively correlated objectives, which lead to the 'anomalous' distributions presented in Fig. 9.7. Another type of anomaly is when

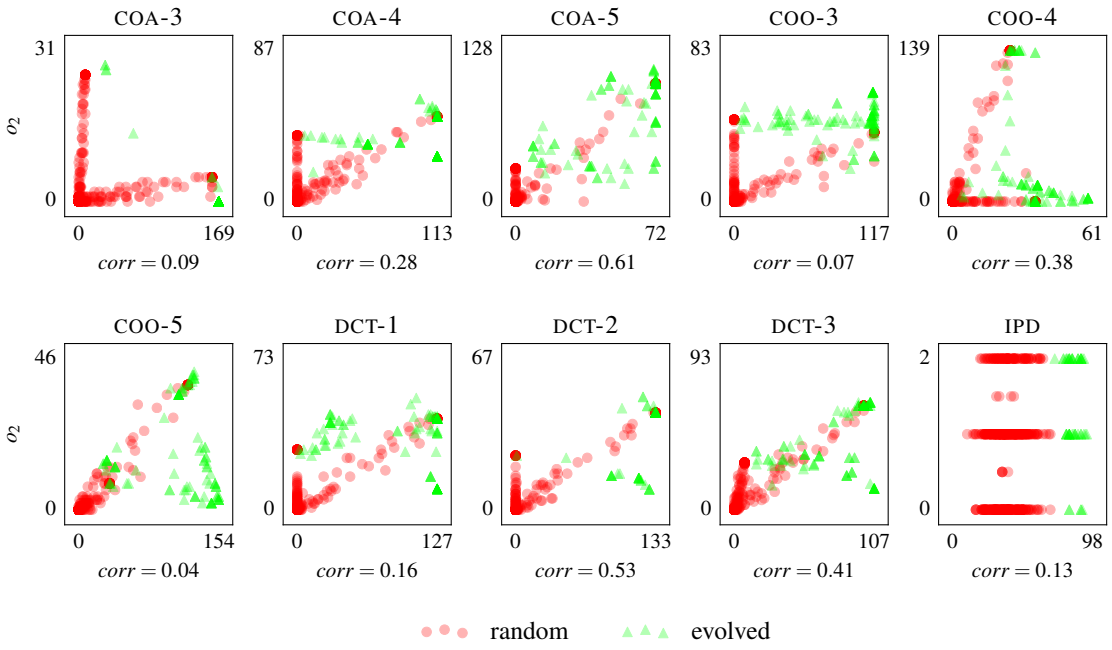


Figure 9.7: Visualization of the positively correlated derived objectives. The evolved candidate solutions marked in green, the random ones in red.

the number of tests that support particular objectives becomes highly imbalanced. In some cases that imbalance may strongly distort the distribution of solutions, as in the case of IPD, where the objective plotted on the ordinate consists of only two tests, causing both random and evolved solutions to align in five horizontal layers, corresponding to the possible aggregated outcomes of interactions $\{0.0, 0.5, 1.0, 1.5, 2.0\}$ with the tests. In the case of COA-3 in Fig. 9.7, we observe dense clusters of both candidate and random solutions near the axes, indicating overspecialization to one of the objectives. In such a case, neither the evolved nor the random solutions trade-off the skills identified by the objectives particularly well.

Let us emphasize, however, that our distinction of 'normal' (Fig. 9.6) and 'anomalous' (Fig. 9.7) behaviors of DOC is rather subjective and intended only to illustrate the spectrum of possible outcomes. In general, we did not observe significant correlation between the 'esthetics' of solutions' arrangements and the performance. Also, these graphs present the state of search objectives in the final population of the run, when good solutions have been usually already found and it becomes difficult to make further progress.

9.5 Experimental analysis in the domain of GP

Program synthesis from examples is considered a particularly demanding class of test-based problems. Recall from Section 5.5.2 that it is commonly approached with GP, where the set of tests T is fixed and given as a part of problem formulation. The search dynamic of GP algorithm is thus entirely different than that of CoEA, where T changes with time. The experiments in this section are intended to demonstrate that DOC proves useful also in such environments.

In the following, we examine the capabilities of DOC by performing its experimental assessment on 17 discrete program synthesis tasks representing two different domains. To a large extent, the experiments conducted here share their configuration with the experiments performed in Sec-

Table 9.3: The common parameter setting for all methods in the experiment.

Parameter	Setting
Population size	1000
Population initialization	Ramped half-and-half
Tournament size	7
Crossover probability	0.9
Mutation probability	0.1
Number of runs	50
Termination condition	200 generations or an ideal solution is found

tion 9.4. In the next two sections, we discuss thus only the methods and benchmarks that are specific for GP (within this thesis) and have not been introduced earlier.

9.5.1 Methods

The compared algorithms implement generational evolutionary algorithm and vary only in the selection procedure. Otherwise, they share the same parameter settings, with the initial population filled with the ramped half-and-half operator, subtree-replacing mutation engaged with probability 0.1 and subtree-swapping crossover engaged with probability 0.9. We run a series of experiments with runs lasting up to 200 generations and with population size $|S| = 1000$. The search process stops when the assumed number of generation elapses or an ideal program is found; the latter case is considered a *success*. For a summary of the parameter settings see Table 9.3.

Based on the experiments with CoEAs performed in Section 9.4.5, we employ here the most efficient variant of DOC, i.e., DOC-AVG (from now on, we refer to it simply as DOC). We confront DOC with several control setups. The first baseline is the conventional Koza-style GP discussed in detail in Chapter 4, which employs conventional scalar evaluation function (6.1.1) as fitness and a tournament of size 7 in the selection phase. The second control is implicit fitness sharing (IFS [241]) presented in Section 8.3.1, with fitness defined as in Formula 8.3.1 and also with tournament of size 7. The last control configuration is RAND, which we introduced in Section 9.4.2.

9.5.2 Benchmark problems

In the form presented in Section 9.1, DOC can handle only binary interaction outcomes, where a candidate solution either passes a test or not. Because of that, we compare here the methods only on problems with discrete interaction outcomes. However, it is a common practice to evaluate GP also on regression problem (cf. Section 5.4.5). Such problems involve a continuous interaction function that typically reflects errors on particular tests. To address them, in Chapter 11 we propose an extension of DOC (and the entire framework for discovery of search objectives) to continuous domains. The effectiveness of DOC on regression problems is discussed in detail in Section 11.2

In Table 9.4, we summarize the benchmark program synthesis task used in this chapter. There are 17 benchmarks in total, 7 of which belong to the Boolean domain and 10 to a categorical domain. The table provides the instruction set used in each domain as well as the number of variables and tests for every problem. The last column shows the size of the search space, which is considered by the compared search algorithms.

Table 9.4: The program synthesis task used in the experiment.

Domain	Instruction set	Problem	Variables	$\|T\|$	Cardinality of search space
Boolean	and, nand, or, nor	Cmp6	6	64	2^{64}
		Cmp8	8	256	2^{256}
		Maj6	6	64	2^{64}
		Maj8	8	256	2^{256}
		Mux6	6	64	2^{64}
		Mux11	11	2048	2^{2048}
		Par5	5	32	2^{32}
Categorical	a_i	Dsc_i	3	27	3^{27}
		Mal_i	3	15	3^{15}

Boolean Benchmarks

The first group of program synthesis tasks are Boolean benchmarks, which employ instruction set $\{and,\ nand,\ or,\ nor\}$ and are defined as follows. For an v-bit comparator $Cmp\,v$, a program is required to return *true* if the $\frac{v}{2}$ least significant input bits encode a number that is smaller than the number represented by the $\frac{v}{2}$ most significant bits. In case of the majority $Maj\,v$ problems, *true* should be returned if more that half of the input variables are *true*. For the multiplexer $Mul\,v$, the state of the addressed input should be returned (6-bit multiplexer uses two inputs to address the remaining four inputs). In the parity $Par\,v$ problems, *true* should be returned only for an odd number of *true* inputs.

Algebra problems

The second group of benchmarks are the algebra problems originating from Spector *et al.*'s work on evolving algebraic terms [335]. These problems dwell in a ternary domain: the admissible values of program inputs and outputs are $\{0, 1, 2\}$. The peculiarity of these problems consists of using only one binary instruction in the programming language, which defines the underlying algebra. Below, we present the semantics of that instruction for the considered five algebras:

a_1	0	1	2
0	2	1	2
1	1	0	0
2	0	0	1

a_2	0	1	2
0	2	0	2
1	1	0	2
2	1	2	1

a_3	0	1	2
0	1	0	1
1	1	2	0
2	0	0	0

a_4	0	1	2
0	1	0	1
1	0	2	0
2	0	1	0

a_5	0	1	2
0	1	0	2
1	1	2	0
2	0	1	0

In the following, the employed algebra is indicated by the suffix the name of term to be evolved. For each of the five algebras, we consider two tasks. In the *discriminator term* tasks (*Dsc* in the following), the goal is to synthesize an expression that accepts three inputs x, y, z and is semantically equivalent to the one shown below:

$$t^A(x, y, z) = \begin{cases} x & if\ x \neq y \\ z & if\ x = y \end{cases} \tag{9.5.1}$$

There are thus $3^3 = 27$ fitness cases in these benchmarks. The second tasks (*Mal*), consists in evolving a so-called *Mal'cev term*, i.e., a ternary term that satisfies the equation:

$$m(x, x, y) = m(y, x, x) = y \tag{9.5.2}$$

This condition specifies the desired program output only for some combinations of inputs: the desired value for $m(x, y, z)$, where x, y, and z are all distinct, is not determined. As a result,

Table 9.5: Average and .95-confidence interval of the best-of-run fitness. Last row presents the averaged ranks of configurations.

Problem	DOC	IFS	RAND	GP
Cmp6	**0.94**	**0.94**	0.62	0.54
Cmp8	**0.52**	0.00	0.04	0.02
Maj6	**0.98**	**0.98**	0.86	0.54
Maj8	**0.00**	**0.00**	**0.00**	**0.00**
Mux6	**1.00**	**1.00**	**1.00**	0.98
Par5	**0.06**	0.00	0.00	0.02
Dsc1	**0.28**	0.24	0.00	0.00
Dsc2	**0.52**	0.46	0.06	0.00
Dsc3	**0.92**	0.84	0.50	0.44
Dsc4	**0.10**	0.00	0.00	0.00
Dsc5	**0.74**	0.28	0.02	0.12
Mal1	**0.98**	0.90	0.78	0.88
Mal2	0.84	**0.90**	0.64	0.00
Mal3	0.96	**1.00**	0.78	0.68
Mal4	**0.54**	0.48	0.26	0.00
Mal5	0.98	0.92	**1.00**	0.88
Rank:	1.406	2.188	2.906	3.500

there are only 15 fitness cases in our *Mal* tasks, the lowest of all considered benchmarks. The motivation for the discriminator and Mal'cev term problems is originally that they're of interest to mathematicians [54]. Here, we chose them as benchmarks because of their difficulty and formal elegance.

9.5.3 Results

We focus on two aspects in the analysis that follows: GP's end-of-run success rate and the size of evolved programs. The first one reflects the end-to-end performance of the proposed method, most relevant from the practical perspective of solving discrete-fitness test-based problems. The second aspect is meant to capture the complexity of candidate solutions in tree-based GP, measured as the number of nodes in their trees. The results presented below are averages over 50 independent runs of evolution, repeated for each combination of method and problem.

Table 9.5 shows the success rates obtained by particular methods on each benchmark, with the best results marked in bold. The last row contains the global rank of a given configuration, obtained by averaging the ranks on individual benchmarks.

The benchmarks vary in the level of difficulty, ranging from very easy ones, which are solved in all runs by all methods (Mux6), to difficult ones, on which even the best-performing configurations barely exceed 6 percent probability of success (Par5). Most importantly, DOC tends to systematically improve success rates compared to ordinary GP, regardless of difficulty, on all benchmarks. It also achieves the best overall average rank of 1.406 and outperforms the other control configurations on 13/16 benchmarks. The second place is taken by IFS with the average rank of 2.188 and the highest success rate on 6/16 benchmarks. RAND and GP perform noticeably worse with the best result on 3/16 and 1/16 benchmarks, respectively.

To statistically evaluate these results, we employ the Friedman's test for multiple achievements of multiple subjects [159], for all configurations presented in the table (i.e., in relation to global average ranks in the bottom row in Table 9.5). The obtained p-value for Friedman test is 3.46×10^{-12}, which strongly indicates that at least one method performs significantly different from the remaining ones. To determine the significantly different pairs, we conduct post-hoc analysis

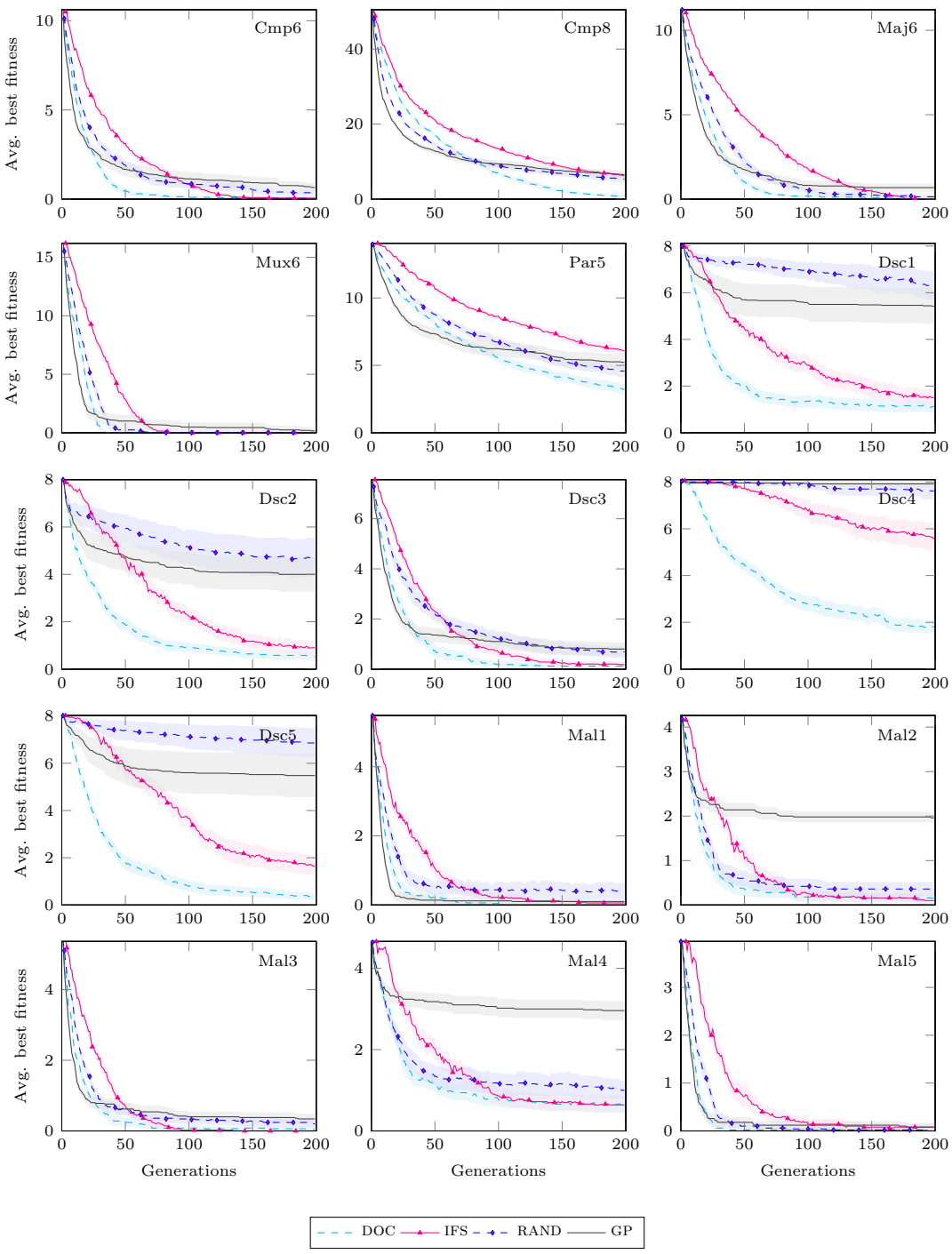


Figure 9.8: Average and .95-confidence interval of the best-of-generation fitness.

Table 9.6: Average and .95-confidence interval of number of nodes in the best-of-run program. Last row presents the averaged ranks of configurations.

Problem	DOC		IFS		RAND		GP	
Cmp6	**133.065**	±16.226	201.650	±12.259	194.589	±21.418	228.660	±23.939
Cmp8	**245.814**	±22.885	268.385	±11.532	249.797	±12.097	314.826	±21.000
Maj6	**189.506**	±17.168	283.115	±12.840	223.874	±20.711	297.198	±21.755
Maj8	**218.388**	±10.376	361.392	±17.457	407.027	±20.506	465.553	±24.007
Mux6	**66.256**	±4.580	148.742	±8.231	88.863	±8.271	113.215	±15.836
Par5	**297.752**	±20.957	299.031	±14.029	337.352	±20.897	407.300	±22.916
Dsc1	158.997	±12.039	165.074	±9.003	**31.701**	±15.222	86.639	±25.317
Dsc2	161.880	±12.989	170.963	±10.687	**86.124**	±23.018	128.936	±21.391
Dsc3	**116.893**	±12.104	179.495	±11.594	162.862	±12.295	164.792	±16.247
Dsc4	194.817	±14.981	56.008	±12.826	10.182	±11.115	**5.599**	±8.840
Dsc5	132.496	±11.019	123.598	±12.441	**23.955**	±12.528	55.948	±21.743
Mal1	92.888	±11.478	135.065	±11.584	95.678	±17.128	**84.096**	±10.078
Mal2	**96.845**	±10.676	132.411	±10.271	102.337	±14.570	160.670	±12.247
Mal3	**111.727**	±12.740	132.598	±8.652	122.724	±17.083	138.464	±16.686
Mal4	158.938	±13.725	162.231	±12.207	**145.837**	±18.957	160.373	±17.089
Mal5	43.431	±4.457	91.444	±10.786	**40.114**	±8.053	49.912	±7.409
Rank:	1.812		3.312		1.875		3.000	

using the symmetry test [135]. Table 9.7 presents the p-values for the hypothesis that a setup in a row is better than a setup in a column (the significant p-values are marked in bold). This comparison reveals that the performance improvement of DOC relative to control methods GP and RAND is significant. The difference is however statistically insignificant for IFS, but DOC achieves higher success rates more often and manages to solve a problem that remained unsolved by other algorithms, i.e., Dsc4.

Figure 9.8 presents the average fitness of the best-of-generation solution (technically, we present the complement of fitness, i.e., $n - f(p)$). The figure demonstrates that in the initial stages of search, when population contains still many random programs, DOC behaves similarly to the control configurations. Its superiority becomes evident in the later stages of evolutionary process, when the programs in the population become more sophisticated and start exhibiting more complex behaviors. Conventional evaluation function in GP often fails to differentiate candidate solutions at this stage, which results in premature converge. This observation is particularly conspicuous in the case of Dsc1 and Mal2 benchmarks. For more difficult problems such as Dsc4, scalar evaluation does not provide useful search gradient, leading to nearly flat curves throughout entire learning process. DOC on the other hand maintains steady progress on most benchmarks and solves on average more tests than the other configurations. This is also true for IFS, however DOC tends to make progress at a faster rate, finding an ideal solution earlier.

The improved performance of DOC can be at least to some extent attributed to its capability to overcome the negative effects of overspecialization on individual tests (cf. Section 3.7). For instance, in the Boolean benchmark *Cmp6*, the task is to determine whether the number encoded by the three least significant input bits b_0, b_1, b_2 is smaller than the number encoded by the three most significant bits b_3, b_4, b_5. A program that checks if b_2 is off and simultaneously b_5 is on solves the quarter of the total of $2^6 = 64$ tests in this task. This can be expressed with a mere few instructions from the assumed instruction set, e.g., as $(b_2\ nor\ b_2)\ and\ b_5$. Conventional scalar evaluation function employed by GP makes it particularly likely for evolution to exploit such opportunities by synthesizing programs that focus on such easy subproblems; however, their further extension to cover the other tests may turn out to be difficult and cause evolution to stall.

Table 9.7: Post-hoc analysis of Friedman's test conduced on Table 9.5 (left) and 9.6 (right): p-values of incorrectly judging a setup in a row to achieve better fitness than a setup in a column. Significant values ($\alpha = 0.05$) are marked in bold.

	DOC	IFS	RAND	GP
DOC		0.253	**0.002**	**0.000**
IFS			0.326	**0.010**
RAND				0.499
GP				

	DOC	IFS	RAND	GP
DOC		**0.006**	0.999	**0.046**
IFS				
RAND		**0.009**		0.066
GP		0.903		

Conversely, DOC attempts to autonomously decompose a problem into a few subproblems (meant as groups of tests) that are meant to prevent such situations. It is likely that at least one derived search objective will correspond to the easier tests, but even if a candidate solutions manages to score perfectly on that objective, it is not guaranteed to survive if it does not make progress on other objectives as well. Moreover, candidate solutions that solve the other, disjoints subsets of tests have also chance to survive in the population, and at some point be recombined with the other solutions, potentially leading to offspring that combines their skills.

Table 9.6 shows the average and 95% confidence interval of the number of nodes in the best-of-run programs. The best-of-run programs produced by DOC turn out to be the smallest on average. To an extent this was expected: DOC achieves the best success rate, and the runs are terminated at success, so the runs of this method last for the lowest number of generations on average. As programs in GP tend to grow with time [198], the best programs found in the early stages of evolution are likely to be smaller than their counterparts in the later stages. However, IFS, which is the runner-up in the ranking in terms of success rate, turns out to produce much larger programs, even though its average run lengths (not reported here) do not diverge much from those of DOC. This observation is confirmed by the Friedman's test shown in Table 9.7 – DOC produces significantly smaller programs than IFS and GP. This may suggest that DOC implicitly promotes more compact solutions. Given that multiobjective selection in DOC requires uniform progress on all derived search objectives in order to make progress, one plausible hypothesis that explains this phenomenon is that overgrown programs fail to generalize beyond certain subsets of tests and are thus discarded during selection. Also, its reasonable to assume that at least one of derived search objectives actually promotes simpler programs.

9.6 Discussion

The experiment demonstrated that DOC is able to identify meaningful search objectives (Section 9.4.9) that are often internally cohesive (Section 9.4.8) and mutually non-redundant (Section 9.4.7). The method autonomously adjusts the number of search objectives to the problem characteristics and the dynamics of evolutionary search (Section 9.4.6), and systematically improves the performance in comparison to the conventional CoEA and GP driven by the scalar evaluation (Sections 9.4.5 and 9.5.3). The method maintains these features across diversified benchmarks of various difficulty.

The search objectives can be arranged into coordinate systems that have natural graphical interpretation (Figs. 9.6 and 9.7). In this respect, they are similar to the coordinate systems of underlying objectives studied in the past works of Bucci and de Jong [35, 69]. The apparent similarity notwithstanding, the derived objectives cannot be however expected to correspond to the underlying objectives, for several reasons. As we have shown in Sec. 9.3, DOC can introduce additional (i.e., not backed up by an interaction matrix) dominance relationships between the

candidate solutions; the coordinate systems of underlying objectives, to the contrary, are *exact* in perfectly preserving the dominance. Secondly, the clustering conducted by DOC is heuristic and thus not guaranteed to optimally assign the tests to the search objectives. Finally, DOC allows the interaction outcomes to be arbitrarily valued, while the exact coordinate systems assume binary interaction outcomes.

The heuristic character of DOC is advantageous in several respects. Firstly, it entails only moderate computational overhead (cf. Section 9.7), while the problem of construction of an exact coordinate system has been proven in [144, 141] to be NP-hard. Secondly, by being based on the outcomes of interactions with a transient population of tests T, the search objectives match the current capabilities of the candidate solutions in S. In other words, the search objectives evolve along the candidate solutions and may adapt to their capabilities, creating a suitable search gradient while avoiding overspecialization.

In the coevolutionary settings, the tests in T are rewarded for distinctions calculated directly from the original interaction matrix G (and not from G'). Therefore, they are not explicitly affected by DOC's multi-objective evaluation. This implies that the first iteration of the evolutionary loop in DOC produces the same second population of tests as in CEL (and in all other configurations considered here). However, the candidate solutions selected in that iteration from S in DOC are likely to be different from those selected in the first iteration of CEL. This results in a different interaction matrix in the second generation, and consequently other selection outcomes in T. In this indirect way, the multi-objective evaluation of DOC affects the dynamics of evolution in the population of tests.

By providing otherwise unavailable grounds for preferring some solutions over the others, the objectives derived by DOC are expected to be more informative than the scalar evaluation. However, one must admit that the process of discovering the objectives is not always guaranteed to succeed. DOC relies heavily on the possibility of identifying certain patterns in interaction outcomes. The more similar the behavior of candidate solutions on the tests is, the harder it becomes for a clustering algorithm to discover the groups of tests that could be attributed to skills. For instance, when disengagement [44] occurs in coevolutionary settings, i.e., all tests currently in T are solved by all candidate solutions (or all are failed), all tests end up in the same cluster, causing the method to degenerate to a single-objective approach and lose its upper hand. Scenarios close to disengagement (a very large fraction of solved or failed tests) may also cause the numbers of tests supporting different objectives to become highly unbalanced, leading to disrupted approximation of the Pareto front. Furthermore, DOC may also underperform when the original objective function is inherently single-objective (for instance in simple problems) or hard to automatically decompose into search objectives. Finally, DOC can discover search objectives only if their existence is manifested *behaviorally*, i.e., reflected in the outcomes of interactions between candidate solutions and tests. If the skills do not manifest in interaction outcomes, DOC has no means to discover them. This may be the case in the problems where passing any test requires all (or most) skills.

9.7 Computational overhead

The process of discovering search objectives obviously incurs an additional computational cost, which for the k-means-like heuristic clustering algorithms is of the order of $O(knm)$, where m and n are respectively the sizes of S and T. The multi-objective selection is also more computationally demanding than the traditional selection operators based on scalar fitness – the complexity of NSGA-II algorithm is $O(nm^2)$ [75]. The total overhead $O(knm+nm^2)$ is thus linear in the function

Table 9.8: Algorithm runtimes (in seconds) for the coevolutionary benchmarks, averaged over 60 evolutionary runs.

	CEL	1-MEANS	RAND	DOC	DOC-BIN	DOC-AVG
IPD	15.7 ± 0.1	14.6 ± 0.2	14.4 ± 0.2	17.2 ± 0.4	17.6 ± 0.2	17.4 ± 0.2
DCT-1	394.7 ± 4.2	428.1 ± 7.1	436.2 ± 6.3	472.4 ± 5.7	450.8 ± 6.3	468.6 ± 6.2
DCT-2	511.2 ± 7.3	526 ± 7.8	529.3 ± 15.7	616.1 ± 12	618.1 ± 16	622.5 ± 16.4
DCT-3	1420.8 ± 48.9	1429.6 ± 45.9	1438.7 ± 58	1424.1 ± 56.4	1483.2 ± 60.6	1471.4 ± 51
COO-3	98.9 ± 5.6	101.9 ± 0.2	104.8 ± 0.2	106.9 ± 5.3	105.7 ± 5.3	105 ± 5.4
COO-4	99.8 ± 5.5	102.9 ± 0.2	105.8 ± 0.2	107.4 ± 4.9	105.9 ± 4.9	104.7 ± 5.2
COO-5	100.2 ± 5.4	104.1 ± 0.1	105.3 ± 0.3	109.2 ± 4.9	107.9 ± 4.5	105 ± 5
COA-3	114.6 ± 5.9	115 ± 0.2	116.9 ± 0.2	116.3 ± 5.3	118.6 ± 5.6	119.8 ± 5.6
COA-4	113.5 ± 5.8	116.2 ± 0.2	115.9 ± 0.2	116.5 ± 5.3	119.9 ± 5.5	116.9 ± 5.3
COA-5	115.1 ± 5.3	114.9 ± 0.2	115 ± 0.3	117.4 ± 5	121.5 ± 4.9	122.5 ± 5

of n, which encourages using DOC with relatively large populations of tests and moderately sized populations of candidate solutions.

These expenses result however from DOC's postprocessing of interaction outcomes, while in many applications it is the interactions that consume the majority of computational budget. This particularly applies to many test-based problems, where there are *multiple* tests to interact with, and a single interaction outcome may require running a possibly complex program (in GP), performing a costly simulation, or playing a game that involves multiple turns. In such cases, the cost of clustering and multi-objective selection may be an insignificant fraction of the overall computation time.

The empirical evidence gathered from our experiments confirms the moderate overhead of the derivation process. Table 9.8 presents the runtimes accompanied by 95% confidence intervals for particular methods and the coevolutionary benchmarks considered in Section 9.4. The times are clearly higher for DOC when compared to CEL across all the benchmarks, but the overhead is only 7.12 percent on average w.r.t. CEL, and it never exceeds 17.03 percent. These numbers could be further reduced by using more efficient algorithms or, e.g., limiting the number of internal iterations of the X-MEANS clustering algorithms (which normally proceeds until datapoints stop migrating between clusters). This would not necessarily deteriorate the quality of evolved solutions, because optimal clustering is probably not essential here, given that the evolutionary search is by nature stochastic. For brevity, we do not report here the runtimes of GP algorithms considered in Section 9.5, however similar observations hold also there.

9.8 Chapter summary

The DOC algorithm described in this chapter is a means to widen the evaluation bottleneck between the fitness function and a search algorithm. By providing the search process with multiple characteristics of candidate solutions, DOC makes a search algorithm better informed.

In replacing the original objective function with heuristic and transient objectives, DOC clearly subscribes to the proposed framework of discovery of search objectives (cf. Section 8.4). Relying on such objectives is not necessarily less efficient than using the standard scalar evaluation function that counts the number of passed tests. In a rugged and multimodal fitness landscape, the original objective may actually turn out to be more deceptive than an imperfect search objective. This becomes particularly true in DOC, where multiple diversified search objectives are used simultaneously and so mitigate premature convergence.

Chapter 10

Discovery of Search Objectives by Factorization

In this chapter we introduce a second method that subscribes to the framework for discovery of search objectives in test-based problems (cf. Section 5.1). The approach, dubbed DOF (Discovery of Search Objectives by Factorization), uses the popular machine learning technique of *non-negative matrix factorization* (NMF) to heuristically derive a low number of search objectives from an interaction matrix between candidate solutions and tests. NMF, which proved effective in, among others, *recommender systems* and multidimensional data analysis, allows us to capture the major factors that characterize interaction outcomes in G. These factors are the primary building blocks of search objectives derived by DOF. We demonstrate that, when employed to drive search, they foster diversification of search directions while maintaining a useful search gradient for the entire evolution.

We begin our considerations by formally introducing the concept of matrix factorization in Section 10.1. Next, in Section 10.2, we describe in detail the proposed approach, and discuss several ways in which factors discovered by NMF can be used to define search objectives. In Section 10.3 and 10.4, we discuss the properties of DOF, and subsequently, in Section 10.5, we present the results of comparative experiment involving, among others DOF and DOC in the domain of tree-based GP. We conclude the chapter with discussion in Section 10.6 and summary in Section 10.7.

The preliminary version of the method presented in this chapter has been published in [213].

10.1 Non-negative Matrix Factorization

Given a non-negative $m \times n$ matrix G and a desired *factorization rank* r, non-negative matrix factorization [21, 112] searches for non-negative matrices (*factors*) W and H that together form a lower rank approximation of G, i.e.:

$$G \approx WH \;\; s.t. \;\; W, H \geq 0, \tag{10.1.1}$$

where $W \in \mathbb{R}^{m \times k}$ is a *weight matrix* and $H \in \mathbb{R}^{k \times n}$ is a *feature matrix* (or a *basis matrix*). In the context of DOF, G is a matrix of interactions between candidate solutions in S and tests in T, and each candidate solution $s \in S$ is associated with a row in W (a vector $w_s \in \mathbb{R}^k$), and each test $t \in T$ corresponds to a column in H (a vector $h_t \in \mathbb{R}^k$), i.e. $m = |S|$ and $n = |T|$. This correspondence inclines us to abuse the notation and index the elements, rows, and columns of matrices with candidate solutions and tests.

Algorithm 6 Two-block coordinate descent framework for NMF.

Require: factorization rank r, population size m, number of tests n, number of steps l.

1: **function** NMF(G)
2: $W, H \leftarrow$ INITIALIZE(m, n, r)
3: **for** $i = 1, 2, \ldots, l$ **do**
4: $W =$ UPDATE(G, H, W)
5: $H =$ UPDATE(G, W, H)
6: **return** W, H

In NMF, the n-dimensional outcome vectors (Eq. 6.4.1) in G are represented in a r-dimensional linear subspace spanned by the basis vectors h_j, $j = 1, \ldots, r$, and their coordinates are given by the vectors w_s. By setting $r \ll min(m, n)$, the high-dimensional data can be represented by a set of low-dimensional vectors in the hope that the basis vectors can discover the latent semantic structure in G. Notice also that if G's rank is $\leq r$, there exists an exact solution to (10.1.1).

To perform the factorization, particularly when $rank(G) > r$, which is the case in DOF, equation (10.1.1) is commonly reformulated as the following optimization problem:

$$\min_{W,H} L(W, H) \equiv \frac{1}{2}||G - WH||_F^2 \;\; s.t. \;\; W, H \geq 0, \tag{10.1.2}$$

where $||\cdot||_F$ is the Frobenius norm. Quantifying the discrepancy between G and its approximation WH with the (quadratic) Frobenius norm is reasonable in many practical situations, as it implicitly assumes the presence of Gaussian noise in the factorized matrix. It allows also to compute the optimal approximation using truncated singular value decomposition, albeit only when strict non-negativity of W and H is not required. As opposed to its unconstrained variant, NMF is NP-hard in general [351]. Most algorithms, including the one used here and explained in the following, resort to standard nonlinear optimization methods, and are only guaranteed to converge to stationary points. On the positive side, these heuristics proved to be successful in many practical applications, and they typically run in $O(mnr)$ operations.

The minimization problem given by (10.1.2) is non-convex when formulated with both W and H simultaneously holding the variables; however it is convex in either W or H. Thus, by keeping one matrix constant, the other can be found by solving simple non-negative least squares problem; for instance, for fixed W, we have to solve

$$\min_{H \geq 0} ||G - WH||_F^2.$$

This is the main reason why virtually all NMF algorithms follow a two-block coordinate descent scheme, shown in Algorithm 6. The update rules in lines 4 and 5 are expected to decrease the approximation error given by (10.1.2), and are typically applied for a fixed number of iterations, or until the error is sufficiently small.

The most straightforward way to initialize W and H in line 2 of Algorithm 6 is to generate them randomly, e.g. by sampling individual weights uniformly from the interval $(0, 1)$. There are several more advanced initialization strategies that aim at reducing the number of iterations necessary to obtain a good factorization, or converging to a better stationary point. However, these techniques typically do not provide any formal guarantees such as the upper bound on the number of steps necessary for convergence, which is actually not surprising, given NP-hardness of NMF.

The most popular approach to implement the UPDATE steps in Algorithm 6 is the multiplicative update algorithm (MU) [207], which alternates the following two steps:

$$w_{sj} \leftarrow w_{sj} \frac{(GH^T)_{sj}}{(WHH^T)_{sj}}, \tag{10.1.3}$$

$$h_{jt} \leftarrow h_{jt} \frac{(W^T G)_{jt}}{(W^T WH)_{jt}}. \tag{10.1.4}$$

In each iteration, the new values of W and H are found by multiplying the current ones by a factor that depends on the quality of approximation. The above update rules are essentially parameter-free and have been shown to monotonically improve NMF's quality of approximation in [207]. As discussed in [48], in order to guarantee convergence to a stationary point, it is important to reinitialize the entries of W that happen to decrease to zero to a small positive value whenever their partial derivatives become negative. The reason for this is that MU can only modify non-zero entries of W. Another possibility is to use a rescaled gradient descent method, see e.g. [17].

As originally proposed by Lee and Seung [207], MU converges rather slowly, see e.g. [119] for a theoretical analysis. The convergence speed can be however significantly improved by updating W several times before updating H because the products HH^T and GH^T do not need to be recomputed [113]. There are also plenty more effective NMF implementations that adhere to Algorithm 6, such as ALS, ANLS, or HALS. For a more in-depth review of these and other algorithms, we refer the reader to [112].

An important difficulty of using NMF in practice is *non-uniqueness* [204], i.e., existence of equivalent factorization outcomes (W', H') such that $WH = W'H'$. It is easy to show that any matrix U such that $WU \geq 0$ and $U^{-1}H \geq 0$ generates an equivalent factorization into $W' = WU$ and $H' = UH$. For instance, U can be chosen as a monomial matrix, i.e. a matrix with exactly one positive entry in each row and column, which results in scaling and permutation of factors in W and H. As it will become apparent soon, such transformations are not an issue in DOF. More problematic are however the non-monomial matrices that satisfy the above conditions because they may lead to factorizations with different interpretations, see e.g. [111]. A common extension that addresses this issue in practice is *regularization* that forces the factors to be *sparse*, i.e. feature relatively few non-zero elements. Sparse representation encodes much of the data using few active components, which makes the encoding easier to interpret. Sparseness in both W and H has also been shown to be crucial for NMF to learn parts-based and intuitive features of data [206]. In practice, sparsity can be enforced by adding a penalty term, such as a L1-norm penalty:

$$\min_{W,H} f(W,H) \equiv \frac{1}{2}||G - WH||_F^2 + \lambda(||W||_1 + ||H||_1), \tag{10.1.5}$$

where λ controls the degree of regularization. We refer the reader to [137] for recent results on the non-uniqueness of NMF.

Much of the appeal of NMF comes from its ability to automatically extract sparse and easily interpretable factors that correspond to *underlying features* of G. The non-negativity constraint allows the model to learn parts-like representations by additively combining features that attempt to 'reproduce' the original input. This characteristic is consistent with many real-world applications, where parts, to form a whole, usually need to be combined additively (and not subtracted) [206]. No wonder then that NMF became a popular tool in pattern recognition or classification, where it shows its strengths in learning meaningful features from real-life datasets such as collections of face images [206] or text documents [274]. Other interesting applications of NMF include air emission control [267], computational biology [77], or blind source separation [45]. In the following, we employ NMF as the core component of our algorithm for discovery of search objectives.

10.2 DOF

DOF employs NMF to extract the factors from G, and subsequently uses them to define transient search objectives. In terms of evolutionary computation, it plays a role of a selection method that subsumes also the process of evaluation. When applied to a population S of m candidate solutions and a set T of n tests, and given a factorization rank r, a parameter of the algorithm, it proceeds as follows (we provide both textual description here, and an algorithmic summary in Algorithm 7):

1. Calculate the $m \times n$ matrix G of interactions between the candidate solutions in S and the tests from T.
2. Perform NMF to factorize G into an $m \times r$ matrix W and $r \times n$ matrix H.
3. Calculate the r derived search objectives d_j, $j = 1, \ldots, r$, defining the score of candidate solution s_i on objective d_j as
$$d_j(s_i) = w_{ij}. \tag{10.2.1}$$
In other words, W is interpreted as a $m \times r$ derived interaction matrix G', with the elements defined as $g'_{ij} = d_j(s_i)$.
4. Apply a multiobjective selection method to the objectives d_j in order to select the parents and generate candidate solutions for the next generation.

In the **DOF-W** variant of the method, the columns of W are treated as search objectives, while their values for candidate solutions are taken directly from the corresponding rows in W (10.2.1). Search objectives for a candidate solution s can be thus interpreted as the coordinates of outcome vectors $o(s)$ in r-dimensional space induced by NMF. Because all elements of G, W and H are non-negative, and multiplication of W and H composes weights and factors additively, the resulting search objectives are guaranteed to contribute only positively to interaction outcomes (or not contribute in rare cases when an element of W or H is strictly zero). This allows us to assume that they are positively correlated with interaction outcomes in G, and treat them as objectives to be maximized in multiobjective selection (step 4).

NMF transforms G into a *joint* latent factor space such that the outcomes of interactions are modeled as inner products in that space. It follows that W and H *together* convey all necessary information that explains the individual outcomes in G. It seems therefore wasteful to deliberately discard the information in H. To address this issue, we propose a second variant of the method, dubbed **DOF-WH**, where step 3 of the above DOF algorithm defines search objectives differently, namely as

$$d_j(s_i) = w_{ij}\mathbf{g}_i^T \cdot \mathbf{h}_j = w_{ij} \sum_{k \in I(s)} g_{ik} h_{jk}, \tag{10.2.2}$$

where $I(s) = \{k : t_k \in T, g(s, t_k) = 1\}$. In this variant (see also Algorithm 7), a search objective weighs the features in H describing the tests $t_k \in T$, $k \in I(s)$ by candidate solution's factors in W. As a result, the jth objective for a given candidate aggregates the jth factor for that candidate in W and the jth factor for all tests in H that were *solved* by this candidate. The motivation for the WH variant is as follows. NMF decomposes individual interaction outcomes in G into the sum of r components, each multiplying a weight in W by a factor in H, i.e.:

$$g_{st} \approx w_{s1}h_{1t} + w_{s2}h_{2t} + \cdots + \underbrace{w_{sj}h_{jt}}_{j\text{th component}} + \cdots + w_{sr}h_{rt}. \tag{10.2.3}$$

The jth search objective derived by DOF-WH can be thus interpreted as the sum of jth components, each being a fraction of the outcome resulting from an interaction of s with $t_k \in T$, $k \in I(s)$. See also Example 10.1 for a more detailed illustration of DOF-WH.

Algorithm 7 Discovery of search objectives via factorization (DOF).

Require: factorization rank r.

```
1:  function DOF(S, T)
2:      for s ∈ S do
3:          for t ∈ T do
4:              g(s,t) ← INTERACT(s,t)          ▷ computes an interaction matrix G
5:      W, H ← NMF(G, r)
6:      for j ∈ 1, ..., r do
7:          for s_i ∈ S do
8:              if DOF-W then                   ▷ determines a variant of the method
9:                  g'_j(s_i) ← w_ij
10:             else
11:                 g'_j(s_i) ← w_ij g_j^T · h_k
12:     return G'
```

The non-negativity constraint motivated us also to change the encoding of interaction outcomes from the conventional 0s and 1s (cf. Section 8.2) to respectively 1s and 2s. Allowing zeroes in G would force the NMF algorithm to approximate them from above only and in general lead to solutions of lower quality, i.e., higher error (10.1.5). With the adopted encoding, the resulting WH matrix is allowed to hold values lower than 1 and greater than 2, and the cases of passing and failing the tests are handled more symmetrically. This also requires adjusting the definition of $I(s)$ accordingly.

Example 10.1. Consider the following interaction matrix G holding the outcomes of interactions with four tests

$$G = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 2 & 2 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{pmatrix} \end{array}, \tag{10.2.4}$$

where the rows correspond to candidate solutions s_1, s_2, and s_3. s_1 passes all tests and has thus fitness of 4, s_2's fitness is 2, and s_3 is the worst with fitness 0. Because the interaction outcomes for the first and the third candidate solution are linearly dependent, this matrix has rank 2 and can be factorized exactly for $r = 2$. The factorization computed by the NMF library in the R package [108] using the `nmf` function applied with default settings is as follows (values rounded to two decimal places):

$$W \times H = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cc} d_1 & d_2 \end{array} \\ \begin{pmatrix} 0.70 & 2.05 \\ 0.73 & 0.66 \\ 0.35 & 1.02 \end{pmatrix} \end{array} \times \begin{array}{c} \\ d_1 \\ d_2 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 0.70 & 0.70 & 2.70 & 2.70 \\ 0.74 & 0.74 & 0.06 & 0.06 \end{pmatrix} \end{array}.$$

In **DOF-W**, the two columns of W form the derived objectives d_1 and d_2. Note that neither of them orders the candidate solutions consistently with fitness: d_1 orders them as follows: s_2, s_1, s_3, d_2 ranks them s_1, s_3, s_2, while the ordering according to the scalar fitness is s_1, s_2, s_3. The derived search objectives are therefore not guaranteed to be consistent with the order imposed by fitness.

This example shows also that DOF-W does not preserve dominance relation (6.2.1) between candidate solutions. Considering our four tests as elementary objectives, the dominances that hold in G are $s_1 \succ s_2$, $s_2 \succ s_3$, and $s_1 \succ s_3$. In the space of the two derived objectives d_1 and d_2, only $s_1 \succ s_3$ holds, and s_1 and s_2 are mutually non-dominated. This was however expected: one

can show formally that reduction of the number of objectives inevitably leads to loss of dominance (see proofs formulated in terms of partially ordered sets (posets) in [141]). Also, preservation of dominance is of secondary importance for DOF: with dozens or more tests used in typical test-based problemns, and diversified candidate solutions' scores on them, the number of pairs of candidate solutions in dominance is very low already in the original space of tests, so there is not much of dominance to be preserved in the NMF process.

Despite not preserving ordering nor dominance, the derived search objectives do convey some information on the structure defined in G. For instance, the average ranks of candidate solutions are 1.5, 2.0, and 2.5, respectively, and they order the candidate solutions consistently with f. Moreover, the derived search objectives convey some preferential information even when $r < rank(G)$. Let us slightly modify G in (10.2.4) by assuming that this time s_3 passed the first test, so that the new interaction matrix is

$$G_1 = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 2 & 2 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 2 & 1 & 1 & 1 \end{pmatrix} \end{array}. \tag{10.2.5}$$

As a consequence, the first and third row of G_1 become linearly independent, and $rank(G_1) = 3$. It is thus impossible to factorize G_1 into such W and H that restore G_1 perfectly for $r = 2$. Indeed, the call of `nmf` function in R package leads to

$$W_1 = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cc} d_1 & d_2 \end{array} \\ \begin{pmatrix} 0.96 & 1.51 \\ 0.39 & 1.84 \\ 0.86 & 0.38 \end{pmatrix} \end{array}, \qquad H_1 = \begin{array}{c} \\ d_1 \\ d_2 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 2.16 & 1.20 & 0.72 & 0.72 \\ 0.05 & 0.35 & 0.90 & 0.90 \end{pmatrix} \end{array}, \tag{10.2.6}$$

the product of which notably diverges from G_2:

$$W_1 \times H_1 = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} 2.17 & 1.70 & 2.07 & 2.07 \\ 0.95 & 1.13 & 1.96 & 1.96 \\ 1.88 & 1.17 & 0.97 & 0.97 \end{pmatrix} \end{array}.$$

However, the average ranks of candidate solutions s_1, s_2 and s_3 are 1.5, 2.0 and 2.5, and thus seem to reflect well their overall performance.

Example 10.2. In order to demonstrate **DOF-WH**, let us now consider the more general case in which an arbitrary interaction matrix G is subject to NMF. Assume $m = 3$, $n = 4$, $r = 2$, and the following matrices W and H resulting from factorization:

$$W = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cc} f_1 & f_2 \end{array} \\ \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} \end{array}, \qquad H = \begin{array}{c} \\ f_1 \\ f_2 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \end{pmatrix} \end{array}.$$

We may now rewrite individual interaction outcomes in G as inner products of the corresponding rows and columns in W and H, i.e.

$$\hat{G} = W \times H = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} w_{11}h_{11} + w_{12}h_{21} & w_{11}h_{12} + w_{12}h_{22} & w_{11}h_{13} + w_{12}h_{23} & w_{11}h_{14} + w_{12}h_{24} \\ w_{21}h_{11} + w_{22}h_{21} & w_{21}h_{12} + w_{22}h_{22} & w_{21}h_{13} + w_{22}h_{23} & w_{21}h_{14} + w_{22}h_{24} \\ w_{31}h_{11} + w_{32}h_{21} & w_{31}h_{12} + w_{32}h_{22} & w_{31}h_{13} + w_{32}h_{23} & w_{31}h_{14} + w_{32}h_{24} \end{pmatrix} \end{array}.$$

The value of search objective d_j for the ith candidate solution is computed by taking a product of the jth weight in the ith row in W and the sum of factors in the jth row in H that correspond to the tests in T solved by the ith candidate solution (Eq. 10.2.2). To make the further part of this example more concrete, let us assume that W and H have been obtained from the interaction matrix G_1, which states that s_1 passes all tests, s_2 passes t_3 and t_4, and s_3 passes only t_1. In that case, the derived interaction matrix G' produced by DOF-WH takes the following form:

$$G' = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cc} d_1 & d_2 \end{array} \\ \begin{pmatrix} w_{11}(h_{11}+h_{12}+h_{13}+h_{14}) & w_{12}(h_{21}+h_{22}+h_{23}+h_{24}) \\ w_{21}(h_{13}+h_{14}) & w_{22}(h_{23}+h_{24}) \\ w_{31}h_{14} & w_{32}h_{24} \end{pmatrix} \end{array}.$$

When individual weights and factors are multiplied, we arrive at the following matrix

$$G' = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cc} d_1 & d_2 \end{array} \\ \begin{pmatrix} w_{11}h_{11}+w_{11}h_{12}+w_{11}h_{13}+w_{11}h_{14} & w_{12}h_{21}+w_{12}h_{22}+w_{12}h_{23}+w_{12}h_{24} \\ w_{21}h_{13}+w_{21}h_{14} & w_{22}h_{23}+w_{22}h_{24} \\ w_{31}h_{11} & w_{32}h_{21} \end{pmatrix} \end{array},$$

It is easy to notice how the derived objectives correspond to the learned components of individual interaction outcomes in $\hat{G}$. For convenience, we rewrite below the relevant parts of $\hat{G}$, highlighting the corresponding components between the matrices:

$$\hat{G} = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} w_{11}h_{11}+w_{12}h_{21} & w_{11}h_{12}+w_{12}h_{22} & w_{11}h_{13}+w_{12}h_{23} & w_{11}h_{14}+w_{12}h_{24} \\ & & w_{21}h_{13}+w_{22}h_{23} & w_{21}h_{14}+w_{22}h_{24} \\ w_{31}h_{11}+w_{32}h_{21} & & & \end{pmatrix} \end{array}.$$

For the candidate solution s_2, the objectives are derived by adding the values in the same colors.

Let us now inspect the objectives derived by DOF-WH from G_1 (10.2.5). When applied for every candidate solution, the Eq. 10.2.2 yields the following search objectives:

$$G'_1 = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cc} d_1 & d_2 \end{array} \\ \begin{pmatrix} 4.61 & 3.32 \\ 0.56 & 3.31 \\ 1.86 & 0.02 \end{pmatrix} \end{array}.$$

Compared to W_1 (10.2.6), the second search objective d_2 now orders the solutions consistently with fitness. This is however not guaranteed to happen in general. Concerning the dominances, only $s_2 \succ s_3$ is absent from G', which suggests that DOF-WH may have the potential to elicit a more useful search gradient than DOF-W. ■

10.3 Properties

DOF is engaged independently in each generation of an evolutionary run. Therefore, the derived objectives are transient, reflecting the contents of interaction matrix at a given stage of search process (cf. Section 8.4.3). In particular, one should not expect the objectives derived in a given generation to correspond in any way to the objectives derived in even the very previous or very next generation.

NMF employed by DOF guarantees invariance to permutation of rows and columns in G, insensitivity to duplicate rows or column in G, and resistance to scaling/translation of its values

(cf. Section 8.4.5). Though the latter may affect the absolute values of factors in W and H, this is not relevant from the perspective of search objectives, which care only about the ordering of candidate solutions.

DOF may involve an arbitrary selection method in step 4. Following the design of DOC, our default choice is NSGA-II algorithm [75]. Recall from Section 8.5.4 that NSGA-II is specific in building a joint ranking of equal numbers of parent and offspring solutions, followed by subsequent removal of the worse half of them. This obviously necessitates that evaluations on all objectives need to be known for both parents and offspring. For consistency with this feature of NSGA-II, we apply DOF to the union of parents' and offspring set, so that the set S that gives rise to the original interaction matrix G in the above algorithm stands for that union. However, the outcomes of parents' interactions with tests are already known, as they have been calculated in the previous generation. Only the offspring candidate solutions have to be applied to tests. Therefore, the computational cost in terms of solution-test interactions is the same for DOF as for standard EAs, i.e., $|T|$ interactions per candidate solution per generation.

The values of the factors resulting from NMF can be arbitrary (yet obviously positive). Therefore, one cannot expect the derived search objectives to have similar magnitudes. This is however not problematic for NSGA-II and most other multiobjective selection methods, as they assume objectives to be defined on ordinal scales and never directly compare them with each other. Also, the crowding distance used in NSGA-II to resolve ties on Pareto-ranks is based on the product of normalized objectives [75], so magnitudes are unproblematic there either.

The factorization rank r determines the number of search objectives and controls the degree to which G is 'compressed' into W and H. DOF is designed for low values of r (in the order of a few), because typical multiobjective selection methods like NSGA-II cannot handle large numbers of objectives. Nevertheless, in Chapter 11, we combine DOF with lexicase selection, and let it discover up to 100 objectives. In the extreme case of $r = 1$, W and H become vectors. If $r < rank(G)$, the compression of G into W and H is inevitably lossy. In practice, the rank of G depends on the diversity of candidate solutions in population on one hand, and on the uniqueness of tests in T on the other. In most cases, one should expect that rank to be very high, sometimes even close to $\min(m, n)$, because any two candidate solutions are likely to yield different interaction outcomes for at least one test and, analogously, any two tests are likely to elicit different response from at least one candidate solution.

Let us also note that search objectives derived by DOF *remain undefined* beyond the set of considered candidate solutions S. In other words, given G, W and H obtained from a set of candidate solutions S, little (if anything) can be said about the values of search objectives for a candidate solution not in S (unless it has the same interaction outcomes as a candidate in S). This limitation is however not an issue for DOF, where generalization beyond S is not required.

10.4 Analysis of dominance relation

In Example 10.1, we demonstrated that DOF does not preserve the dominance relation between candidate solutions in G. To be precise, it may commit false negative errors (cf. Section 9.3). In the following, we prove that DOF does not commit arguably the most severe mistake of inverting the dominance relation between candidate solutions in G. We will also prove that DOF cannot commit false positive errors. We start with proving these properties for DOF-W, and then generalize them to DOF-WH.

Definition 10.3. The candidate solutions s_i, s_j are *equivalent* in G ($s_i \sim_G s_j$) if $g_{ik} = g_{jk}$ for every $k = 1, \ldots, n$.

Definition 10.4. The candidate solutions s_i, s_j are *comparable* in G if either $s_i \succ_G s_j$, $s_j \succ_G s_i$ or $s_i \sim_G s_j$.

From these definitions it follows that s_i, s_j are comparable if and only if

$$\forall_k \, g_{ik} \geq g_{jk} \text{ or } \forall_k \, g_{jk} \geq g_{ik},$$

where $k = 1, \ldots, n$.

Assume further that NMF is is applied to G and yields an exact decomposition/factorization, i.e. $G = WH$ (cf. (10.1.1)). For reader's convenience, we detail W and H below:

$$W = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1k'} & \cdots & w_{1r} \\ w_{21} & w_{22} & \cdots & w_{2k'} & \cdots & w_{2r} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{j1} & w_{j2} & \cdots & w_{jk'} & \cdots & w_{jr} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mk'} & \cdots & w_{mr} \end{pmatrix} H = \begin{pmatrix} h_{11} & h_{12} & \cdots & h_{1k} & \cdots & h_{1n} \\ h_{21} & h_{22} & \cdots & h_{2k} & \cdots & h_{2n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ h_{k'1} & h_{k'2} & \cdots & h_{k'k} & \cdots & h_{k'n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ h_{r1} & h_{r2} & \cdots & h_{rk} & \cdots & h_{rn} \end{pmatrix}.$$

Theorem 10.5. *Let G be a non-negative interaction matrix and $G = WH$ be its exact non-negative factorization for some r. Let $s_i \succ_G s_j$ for some i and j, $i, j \in \{1, \ldots, m\}$. If s_i and s_j are comparable in W, then $s_i \succ_W s_j$.*

Proof. From $s_i \succ_G s_j$ it follows there exists a test $t_k \in T$ such that

$$g_{ik} > g_{jk} \tag{10.4.1}$$

for some $k = 1, \ldots, n$. Suppose, for the sake of contradiction, that s_i and s_j are comparable in W and that $s_i \succ_W s_j$ is not true. Then either $s_j \succ_W s_i$ or $s_j \sim_W s_i$. In the first case, there must exist $k' = 1, \ldots, r$ such that

$$w_{jk'} > w_{ik'} \wedge w_{jl} \geq w_{il}, \quad l \neq k', l = 1, \ldots, r. \tag{10.4.2}$$

When these elements of W become multiplied by the elements of H, we consider two subcasess.In the subcase $h_{k'k} > 0$, then we have

$$w_{jk'} h_{k'k} > w_{ik'} h_{k'k}.$$

Hence

$$\underbrace{w_{jk'} h_{k'k} + \sum_{l=1, l \neq k'}^{r} w_{jl} h_{lk}}_{= g_{jk}} > w_{ik'} h_{k'k} + \sum_{l=1, l \neq k'}^{r} w_{jl} h_{lk}$$

$$g_{jk} + \sum_{l=1, l \neq k'}^{r} w_{il} h_{lk} > \underbrace{w_{ik'} h_{k'k} + \sum_{l=1, l \neq k'}^{r} w_{il} h_{lk}}_{= g_{ik}} + \sum_{l=1, l \neq k'}^{r} w_{jl} h_{lk}$$

$$g_{jk} + \sum_{l=1, l \neq k'}^{r} w_{il} h_{lk} > g_{ik} + \sum_{l=1, l \neq k'}^{r} w_{jl} h_{lk}.$$

From the last inequality it follows immediately

$$\sum_{l=1,l\neq k'}^{r}(w_{il}-w_{jl})h_{lk} > g_{ik}-g_{jk}.$$

From $w_{jl} \geq w_{il}$ for $l \neq k'$, $l = 1,\ldots,r$ we have that the left-hand side of this formula must be non-positive. According to (10.4.1), $g_{ik} - g_{jk} > 0$, hence we arrive at contradiction.

The subcase $h_{k'k} = 0$ also leads to contradiction with (10.4.1), because taking into account (10.4.2) we obtain

$$g_{jk} = \sum_{l=1,l\neq k'}^{r} w_{jl}h_{lk} \geq \sum_{l=1,l\neq k'}^{r} w_{il}h_{lk} = g_{ik}.$$

In the second case, i.e. $s_j \sim_W s_i$, we have $w_{jk} = w_{ik}$ for every $k = 1,\ldots,n$. This implies $g_{jk} = g_{ik}$ for every k, hence we arrive at a contradiction. □

Since we have arrived at a contradiction in all cases, our original supposition that $s_j \succ_W s_i$ cannot be true, which proves that the inversion of dominance cannot happen whenever s_i and s_j are comparable in W. Notice also that if s_i and s_j are not comparable in W, then by definition, inversion of dominance in W cannot happen. We summarize this result in a more general way in the following theorem:

Theorem 10.6. *Discovery of search objectives by factorization does not lead to inversion of the dominance relation in G'.*

The above theorem applies instantly to DOF-W. In case of DOF-WH, recall that a factor w_{jk} is multiplied by the expression $c_{jk} = \mathbf{g}_j^T \cdot \mathbf{h}_k$ (Eq. 10.2.2) that is guaranteed to be non-negative and its magnitude depends on the number of solved tests in G by the candidate solution s_j. Notice further that if $s_j \succ_G s_i$, then

$$c_{jk} \geq c_{ik} \tag{10.4.3}$$

because s_j solves more tests than s_i. From (10.4.3) and from (10.4.2) it follows that

$$\begin{aligned} w_{jk'}c_{jk'} &> w_{ik'}c_{ik'} \text{ for } k' = 1,\ldots,r, \\ w_{jl}c_{jl} &\geq w_{il}c_{il} \quad \text{for } l \neq k' \end{aligned}$$

From this point, a similar proof by contradiction can be easily performed for DOF-WH.

In the following, we prove that DOF cannot commit false positive errors.

Proposition 10.7. *Let s_i, s_j be arbitrary candidate solutions. If s_i, s_j are comparable in W, then s_i, s_j are comparable in G.*

Proof. We have $s_i \succ_W s_j$ or $s_j \succ_W s_i$ or $s_i \sim_W s_j$. Let us suppose first that $s_i \succ_W s_j$. Then there exists k', $1 \leq k' \leq r$ such that

$$w_{ik'} > w_{jk'} \wedge w_{il} \geq w_{jl}, \quad l \neq k', l = 1,\ldots,r. \tag{10.4.4}$$

For any fixed k, $k = 1,\ldots,n$ we have

$$g_{ik} = \sum_{l=1}^{r} w_{il}h_{lk} \geq \sum_{l=1}^{r} w_{jl}h_{lk} = g_{jk}.$$

If $h_{k'k} \neq 0$, then $g_{ik} > g_{jk}$ holds, so s_i, s_j are comparable in G. Arguing in a similar manner in both cases $s_j \succ_{G'} s_i$ and $s_i \sim_W s_j$, we obtain, respectively, $g_{jk} \geq g_{ik}$ and $g_{ik} = g_{jk}$. Thus, in any case we have that s_i, s_j are comparable in G. □

From Proposition 10.7 it immediately follows

Theorem 10.8. *If s_i, s_j are incomparable in G, then s_i, s_j are incomparable in W.*

Theorem 10.8 implies that DOF cannot commit false positive errors.

To summarize our considerations, discovery of search objectives by factorization, when performed in exact manner (i.e., $G = WH$), may lead to the loss of dominance (false negative errors) in G' (see Example 10.1), but it is guaranteed not to posit a dominance between candidate solutions if it was not present in G (Theorem 10.8). Finally, the dominance relation between any pair of candidate solutions comparable in G' is never reversed (Theorem 10.6).

It is important to emphasize again that the above theoretical results hold only for the scenarios in which G is factorized into W and H *exactly*, which is the case if $rank(G) \leq r$. No such guarantees exist otherwise. The likelihood of dominance inversions or new dominances arising in the space of derived objectives depends on the characteristics of the heuristic algorithm used for NMF (e.g., the MU algorithm used in this thesis). Nevertheless, as argued in other parts of this chapter, one can assume that the impact of such undesirable phenomena on DOF's performance is likely to be low, given the stochastic character of evolutionary search.

10.5 Experimental evaluation

In this section we present the results of a comparative experiment involving two variants of DOF introduced in Section 10.2. Our goal, beyond demonstrating the strength of factorization-based evaluation (Sections 10.5.2 and 10.5.3), is to answer the following questions:

1. Is the success of methods that derive search objectives more due to diversification, or due to the useful search gradient created by the derived objectives? The experiment addressing this question is reported in Section 10.5.4.

2. How do derived search objectives relate to fitness? What is the dynamics of search process driven by such objectives? In Section 10.5.5, we visualize the objectives derived by DOF and attempt to answer these questions.

We close this section by discussing computational efficiency of DOF methods in Section 10.5.6.

10.5.1 Methods and benchmarks

In the following computational experiments, we assess the characteristics of DOF using as the baseline the standard Koza-style tree-based GP algorithm, which is driven by conventional evaluation function (6.1.1), and IFS with fitness defined as in (8.3.1). We also compare DOF to DOC described in Section 9.1, which derives search objectives by clustering of interaction outcomes.

The factorization of interaction matrix in DOF is realized by the MU algorithm ((10.1.3) and (10.1.4)), implemented as discussed in Section 10.1. To speed-up convergence, we conduct 5 updates of W before updating H. When invoked for a given interaction matrix G, we perform up to 50 iterations of MU. If the approximation error (10.1.2) drops below 10^{-5}, we stop the optimization earlier. We consider three values of the factorization rank $r \in \{2, 3, 4\}$. This choice is dictated by the limited capability of handling larger numbers of objectives by the NSGA-II selection, which we apply to derived objectives in order to perform multiobjective selection of parent solutions for the next generation in DOF.

In order to make the following experiments comparable with those already conducted in Section 9.5, we strictly follow the experimental protocol and the benchmark problems used there. For an overview of the parameters used in evolution, see Table 9.3. The benchmark problems represent

Table 10.1: Success rate of the best-of-run individuals (per 50 runs) and average ranks of methods w.r.t. success rate. Bold font marks the best result for each benchmark.

	GP	IFS	DOC		DOF-W			DOF-WH		
Benchmark				r	2	3	4	2	3	4
Cmp6	0.54	0.94	0.94		**1.00**	0.00	0.00	0.98	**1.00**	0.06
Cmp8	0.02	0.00	**0.52**		0.08	0.00	0.00	0.06	0.00	0.00
Maj6	0.54	0.98	0.98		0.90	0.00	0.00	0.98	**1.00**	0.00
Maj8	0.00	0.00	0.00		0.00	0.00	0.00	0.00	0.00	0.00
Mux6	0.98	**1.00**	**1.00**		**1.00**	0.72	0.00	**1.00**	**1.00**	0.98
Par5	0.02	0.00	**0.06**		**0.06**	0.00	0.00	**0.06**	**0.06**	0.00
Dsc1	0.00	0.24	0.28		0.30	0.20	0.00	0.20	0.38	**0.70**
Dsc2	0.00	0.46	0.52		0.40	0.46	0.00	0.42	0.60	**0.86**
Dsc3	0.44	0.84	0.92		0.88	0.58	0.00	0.86	**0.98**	**0.98**
Dsc4	0.00	0.00	**0.10**		0.00	0.00	0.00	0.00	**0.10**	0.06
Dsc5	0.12	0.28	0.74		0.26	0.36	0.00	0.56	**0.88**	0.78
Mal1	0.88	0.90	0.98		**1.00**	**1.00**	0.98	**1.00**	**1.00**	**1.00**
Mal2	0.00	0.90	0.84		**1.00**	**1.00**	0.92	0.92	**1.00**	**1.00**
Mal3	0.68	**1.00**	0.96		0.90	**1.00**	0.32	0.96	**1.00**	**1.00**
Mal4	0.00	0.48	0.54		0.62	**0.94**	0.74	0.48	0.84	0.86
Mal5	0.88	0.92	0.98		**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rank	6.88	5.47	4.26		4.32	5.44	7.12	4.59	2.85	4.06

the Boolean and categorical domains, and are described in Section 9.5.2. See also Table 9.4 for the details regarding the instruction set, the number of variables and the tests used in each domain.

The results presented below are averages over 50 independent runs of evolution, repeated for each combination of method and problem.

10.5.2 Success rate

The primary performance indicator we consider is success rate, i.e., the percentage of runs that ended up with a correct program. Table 10.1 presents this performance measure for particular configurations and benchmarks. In order to ease the analysis and take into account the fact that some benchmarks are inherently more difficult than others, we summarize Table 10.1 by ranking the methods on every benchmark independently, and reporting the averaged ranks at the bottom of the table. The best results on individual benchmarks are also marked in bold.

The results reveal that the conventional single-objective GP tends to solve reliably only the easiest problems (e.g. Mux6 with the average success rate among all configurations of 85.3). IFS systematically improves on GP's performance, achieving higher success rates on all benchmarks. However, it is the multiobjective methods based on two or more search objectives that deliver the largest leap in performance. Table 10.1 reports the results obtained by six DOF configurations in total, one for each combination of the method's variant and factorization rank r. Of these configurations, DOF-WH with $r = 3$ achieves the best results, having at least as high success rate as the other methods for all problems except for Mal4, Dsc2, and Cmp8. It also appears, that $r = 3$ is the optimal setting for DOF-WH as its overall outcomes for $r = 2$ and $r = 4$ are slightly worse. Judging from the performance on individual benchmarks, DOF-WH with $r = 4$ favors categorical problems, as it systematically outperforms the other methods, except for Dsc5, where it ranks second.

DOF-W, on the other hand, performs the best when $r = 2$. For $r = 3$, we observe evident deterioration of success rates, particularly for Boolean benchmarks. For instance, Cmp6 and Maj6 are not solved even once, whereas the probability of success when $r = 2$ is equal or above 90. This

Table 10.2: Post-hoc analysis of Friedman's test conduced on ranks achieved by the best performing configurations from Table 10.1. Significant values ($\alpha = 0.05$) are marked in bold.

	DOC	GP	IFS	DOF
DOC		**0.001**	0.329	
GP				
IFS		0.167		
DOF	0.369	**0.000**	**0.005**	

trend continues for $r = 4$, for which DOF-W fails to solve most of the benchmarks and ranks as the last configuration. This may suggest that as the factorization ranks increases, the weights in W alone do not differentiate the candidate solutions well enough to create a useful search gradient. Another plausible explanation of this phenomenon is that NSGA-II, which is employed as the selection method in DOF, fails to obtain a representative, evenly distributed approximation of the Pareto front, which is necessary to find good, diversified solutions. Nevertheless, in its best setting, DOF-W takes the 4th spot among all the methods, and manages to rank before GP and IFS. We hypothesize that when $r = 2$ or $r = 3$, NMF is encouraged to come up with factors that model only the critical aspects of the interactions that prove to be particularly suitable to guide the search process.

The performance on individual benchmarks, although interesting, does not reveal much regarding the relations of compared methods in terms of the likelihoods of synthesizing a correct program for any problem. For this reason, in the following, we rank only the best performing DOF configuration (the WH variant with $r = 3$) and the control methods. For clarity of presentation, from now on, by DOF we mean its best performing configuration. The ranks are as follows:

DOF	DOC	IFS	GP
1.406	2.094	2.812	3.688

To statistically evaluate these results, we apply the Friedman's test for multiple achievements of multiple subjects [159] to the above configurations in relation to their average ranks. The obtained p-value for Friedman test is 5.07×10^{-7}, which strongly indicates that at least one method performs significantly different from the remaining ones. To determine the significantly different pairs, we conduct post-hoc analysis using the symmetry test [135]. Table 10.2 presents the p-values for the hypothesis that a setup in a row is better than a setup in a column (the significant p-values are marked in bold). This comparison reveals that the performance improvement of DOF relative to control methods GP and IFS is significant. The difference is however statistically insignificant for DOC. Nevertheless, it is interesting to note that DOF achieves higher or equal success rates on all benchmarks except for Cmp8, which remains unsolved by DOF (cf. Table 10.1).

To analyze the search dynamics of DOF, in Fig. 10.1 we plot the average best-of-generation fitness graphs for particular methods and benchmark problems, with 95% confidence intervals marked as semi-transparent bands. This figure clearly shows that both DOF and DOC tend to improve the learning speed compared to the standard GP algorithm and IFS. In many cases, they not only learn faster, but also achieve a lower values of fitness, which indicates that they solve more tests on average (recall that we technically present the complement of fitness, i.e., $n - f(p)$). When it comes to head-to-head comparison between DOF and DOC, it appears that DOF maintains an upper hand when solving categorical problems. Although the differences are not large, particularly at the beginning of evolution, where both methods make roughly the same progress, they become more evident typically after 50 generations, when DOF's curve diverges from that of DOC and ultimately converges to a lower values. This is particularly easy to notice for Dsc1 and Mal4.

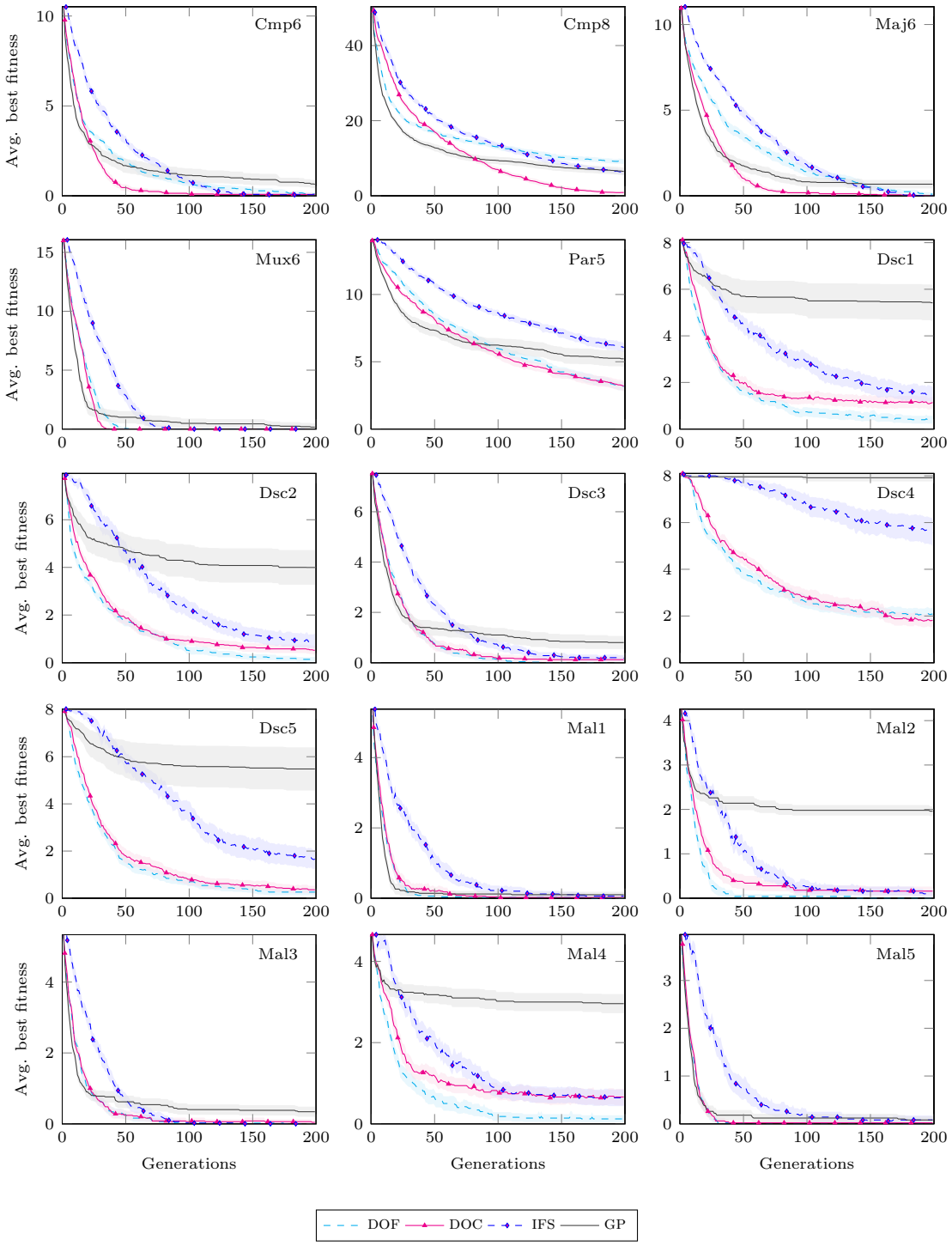


Figure 10.1: Average and .95-confidence interval of the best-of-generation fitness.

Table 10.3: Average and .95-confidence interval of number of nodes in the best-of-run program. Last row presents the averaged ranks of configurations.

Benchmark	DOF		DOC		IFS		GP	
Cmp6	**114.933**	±6.637	133.065	±16.226	201.650	±12.259	228.660	±23.939
Cmp8	**165.444**	±8.203	245.814	±22.885	268.385	±11.532	314.826	±21.000
Maj6	**145.381**	±8.025	189.506	±17.168	283.115	±12.840	297.198	±21.755
Maj8	275.567	±12.806	**218.388**	±10.376	361.392	±17.457	465.553	±24.007
Mux6	67.896	±4.214	**66.256**	±4.580	148.742	±8.231	113.215	±15.836
Par5	**275.617**	±12.527	297.752	±20.957	299.031	±14.029	407.300	±22.916
Dsc1	131.777	±7.187	158.997	±12.039	165.074	±9.003	**86.639**	±25.317
Dsc2	**124.538**	±9.115	161.880	±12.989	170.963	±10.687	128.936	±21.391
Dsc3	**92.281**	±7.806	116.893	±12.104	179.495	±11.594	164.792	±16.247
Dsc4	158.406	±8.016	194.817	±14.981	56.008	±12.826	**5.599**	±8.840
Dsc5	101.505	±6.710	132.496	±11.019	123.598	±12.441	**55.948**	±21.743
Mal1	**63.182**	±4.039	92.888	±11.478	135.065	±11.584	84.096	±10.078
Mal2	**61.033**	±4.452	96.845	±10.676	132.411	±10.271	160.670	±12.247
Mal3	**64.470**	±4.229	111.727	±12.740	132.598	±8.652	138.464	±16.686
Mal4	**87.992**	±8.692	158.938	±13.725	162.231	±12.207	160.373	±17.089
Mal5	45.506	±2.718	**43.431**	±4.457	91.444	±10.786	49.912	±7.409
Rank:	1.438		2.250		3.375		2.938	

Table 10.4: Post-hoc analysis of Friedman's test conduced on Table 10.3. Significant values ($\alpha = 0.05$) are marked in bold.

	DOF	DOC	IFS	GP
DOF		**0.000**	**0.005**	0.283
DOC		0.066	0.434	
IFS				
GP		0.773		

DOC, on the other hand, fares better on Boolean benchmarks, where it tends to outperform DOF in terms convergence rate. These observations are consistent with Table 10.1, which points to the similar conclusions.

10.5.3 Program size

In this section, we compare the methods in terms of complexity of candidate solutions they produce, measured as the number of nodes in their program trees. Table 10.3 shows the average and 95% confidence interval of the number of nodes in the best-of-run programs. For a more detailed insight, we also summarize Table 10.3 by ranking the methods on every benchmark independently, and then averaging the ranks for each method.

DOF produces the smallest programs on 10/16 benchmarks and achieves the average rank of 1.438. It is an impressive feat, given that the programs produced by DOC are already significantly smaller than those of GP and IFS (cf. Table 9.6). DOC ranks second with the average rank of 2.25, outperforming the other methods on 3/16 benchmarks. GP ranks third with the result of 2.938 and, similarly to DOC, obtains the best result on 3/16 benchmarks. IFS appears to be the most susceptible to bloat, producing noticeable more complex candidate solutions, and takes the last spot in the comparison with the average rank of 3.375. The outcomes of the Friedman's test shown in Table 10.4 confirm these observations — DOF produces significantly smaller trees than GP and IFS. The difference between DOF and DOC is however not statistically significant, however the results are clearly in favor of DOF. As suggested by the *Minimum Description Length*

Table 10.5: Average number of generations before an ideal program is found. Bold font points to a configuration that finds it faster.

Problem	DOF		DOC	
Cmp6	118.070	±12.936	**51.000**	±6.330
Maj6	139.978	±9.343	**71.000**	±9.438
Mux6	34.340	±2.217	**26.660**	±1.284
Par5	186.721	±13.398	**143.000**	±9.585
Dsc1	98.676	±11.242	**87.571**	±17.727
Dsc2	104.295	±9.343	**103.462**	±17.691
Dsc3	67.460	±7.858	**57.870**	±8.874
Dsc4	**133.500**	±24.990	139.600	±30.138
Dsc5	**95.846**	±12.006	96.000	±14.056
Mal1	**26.180**	±6.483	29.224	±6.026
Mal2	**28.160**	±6.434	37.167	±9.269
Mal3	40.180	±7.035	**39.104**	±7.741
Mal4	**58.881**	±10.802	70.815	±16.968
Mal5	19.740	±1.693	**18.510**	±2.095

principle [301], DOF's capability to evolve smaller programs may be of crucial importance for its good performance.

Given the above outcome, it is particularly interesting to further scrutinize the results obtained by DOF and DOC. Recall from Section 9.5.3 that without any direct countermeasure, programs in GP tend to bloat with time. We ask thus whether the tendency to produce small programs in DOF stems from the shorter evolution. To this aim, we inspect the average run length of both methods, i.e. the average number of generations before an ideal candidate solution is found, and report the results in Table 10.5. We omit any benchmarks which were not solved at least once by either method, i.e. Cmp8 and Maj8. Interestingly, the results suggest that DOC tends to find the correct programs faster, despite DOF's higher success rates. Indeed, DOC finishes evolution with success earlier than DOF on 9/14 benchmarks, however it is DOF that produces smaller programs on 7 of these (Cmp6, Maj6, Par5, Dsc1, Dsc2, Dsc3, Mal3, cf. Table 10.3). The only exception is Mal5, where DOC both finishes earlier and finds a smaller solution.

As demonstrated here, smaller number of generations indeed positively correlates with delivering of shorter programs. However, it is apparently not the only factor responsible for the synthesis of more concise programs in DOF. Another plausible explanation is that DOF operates on a more syntactically diversified population that leads to reduced bloat [3]. In order to verify the above hypothesis, we measure syntactic entropy of population of candidate solutions. In information theory, the concept of entropy is commonly used to quantify the uncertainty associated with a random variable. More specifically, the entropy $H(X)$ of a random variable X with probability mass function $p(x)$ can be used to measure the average information content that is missing when the value of X is unknown [321]. The entropy for discrete variables, as defined by Shannon, is as follows:

$$H(X) = -\sum_i p_i \log_2 p_i, \tag{10.5.1}$$

where p_i is the empirical chance of observing value i. Crucially for our considerations, a random variable that may take on a large number of values at comparable probabilities has a higher entropy than a random variable with a limited range of values. The random variable X is a function defined on a population S and assuming values in the set $\{c_1, c_2, \ldots, c_l\}$, where c_i counts the occurrences of each unique program tree (expression represented by a tree) in S, i.e $X(s) = c_i$

Table 10.6: Average syntactic entropy of candidate solutions from the last generation of evolutionary run, accompanied by .95-confidence interval.

Problem	DOF		DOC	
Cmp6	**10.665**	±0.012	10.582	±0.033
Cmp8	**10.686**	±0.015	10.554	±0.025
Maj6	**10.661**	±0.016	10.567	±0.033
Maj8	**10.756**	±0.009	10.437	±0.019
Mux6	**10.599**	±0.012	10.490	±0.019
Par5	10.526	±0.013	**10.568**	±0.029
Dsc1	**10.067**	±0.040	9.784	±0.020
Dsc2	**10.046**	±0.051	9.789	±0.013
Dsc3	**10.364**	±0.016	9.844	±0.008
Dsc4	**10.056**	±0.037	9.746	±0.029
Dsc5	**9.752**	±0.058	9.696	±0.026
Mal1	**9.936**	±0.041	9.742	±0.058
Mal2	9.807	±0.050	**9.807**	±0.017
Mal3	**10.313**	±0.032	9.865	±0.008
Mal4	**9.825**	±0.115	9.762	±0.069
Mal5	**9.668**	±0.075	9.537	±0.080

and $P(X = c_i) = \frac{c_i}{|S|}$ for $i = 1, 2, \ldots, l$. In this sense, syntactic entropy is a measure of expected programs' uniqueness in S.

Table 10.6 presents the average syntactic entropy of candidate solutions, accompanied by 95% confidence interval, measured in the last generation of an evolutionary run. Though the absolute differences between DOC and DOF are not big, the trend is clear. DOF tends to maintain higher syntactic entropy on the majority of benchmark problems, which gives us some evidence that expressions (programs) evolved by DOF are characterized by the greater degree of variability on the level of their source code. With more syntactic diversity, it becomes more likely that some of the programs in the population continue improving their fitness. This, in turn, reduces the risk of bloat, i.e. growth of program size without accompanying improvement of fitness.

10.5.4 Behavioral diversity and search gradient

The results presented in Section 10.1 raise the question why DOF performs significantly better than regular GP or IFS. We already showed in the previous section that driving search with multiple derived objectives allows maintaining higher level of syntactical diversity. Another hypothesis is that the induced by DOF diversity is not only syntactical, but also *behavioral (semantic)*. Yet another possibility is that search objectives derived by DOF elicit a more useful search gradient that paves the way for finding an optimal solution faster and more reliably. In particular, given that r is typically much smaller than the rank of the original interaction matrix G, we expect the dominance relation induced by G' to be more dense than the dominance relation in G . In the following, we evaluate both these hypotheses, i.e. we investigate first whether DOF maintains a higher level of behavioral diversity than the other methods, and then we empirically quantify the amount of dominance in G and G' to assess whether there is indeed grounds for claiming that search objectives elicit a more useful search gradient.

Figure 10.2 shows the behavioral diversity of candidate solutions at each generation, measured as the fraction of unique semantics (vector of outputs, Eq. 6.4.3) in the population, and plotted as an average across 50 evolutionary runs. For Boolean benchmarks, DOF maintains the highest level of behavioral diversity over the course of evolution, which exceeds even IFS, which is explicitly designed to increase diversity by reducing the influence of a test on a candidate solutions' fitness

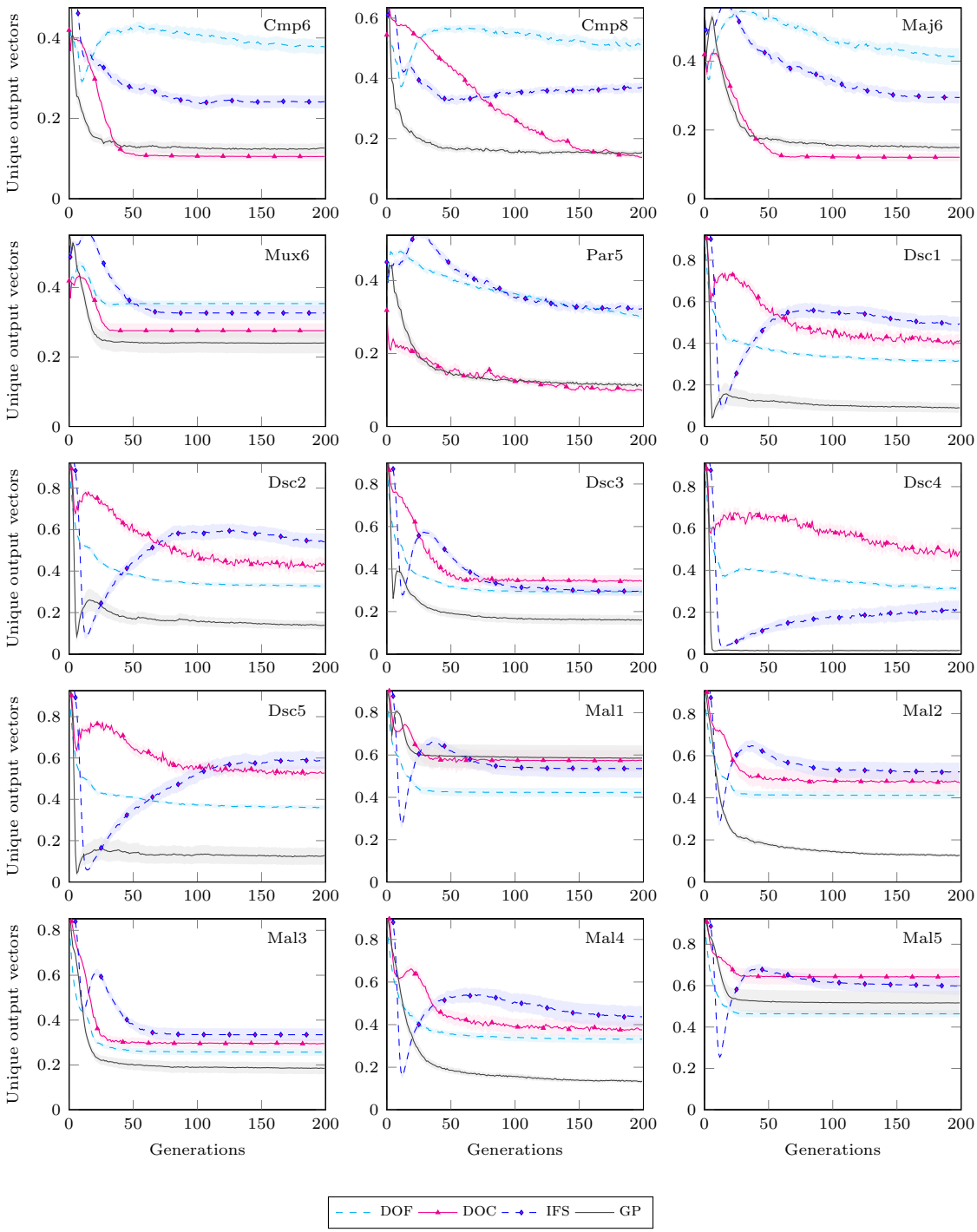


Figure 10.2: Behavioral diversity of candidate solutions at each generation, averaged over 50 evolutionary runs, with 95% confidence intervals marked as semi-transparent bands.

proportionately with the number of candidate solutions in the current population that solve it (8.3.1). The only exception is Par5, where shortly after the initial spike in diversity of IFS between generations 10 and 90, both methods maintain very similar level of diversity. Quite surprisingly, the level of population diversity in DOC drops quickly after the first few generations, sometimes even below the diversity of ordinary GP, and then settles around 0.2. However, given the overall good performance of DOC on Boolean problems (cf. Table 10.1) and its tendency to find optimal solutions early (cf. Table 10.5), we believe this result could be attributed to fast convergence of DOC on these problems. For instance, on Cmp6, DOC takes 51 generations on average to find a perfect candidate solution, while its diversity curve drops from 0.4 to around 0.15 at roughly generation 50, which indeed suggests that the population may have converged.

The plots for categorical problems demonstrate that the level of diversity maintained by DOF drops rapidly for the first 20 generations and then stabilizes for the remaining part of evolution. Though it never falls below 0.3, both DOC and IFS systematically achieve higher levels of behavioral diversity, despite some swings up and down early in the evolution. Interestingly, at the beginning of evolution, IFS behaves similarly to DOF, i.e the diversity dramatically drops, sometimes even below the level of 0.2, and then starts to slowly increase, achieving the highest diversity of all methods on 6/10 problems (Dsc1, Dsc2, Dsc5, Mal2, Mal3, Mal4). DOC, on the other hand, also loses diversity early on, but at a much slower rate, and eventually settles between 0.5 and 0.6. This makes DOC the runner-up in terms of diversity on these benchmarks. On the remaining ones, it achieves the highest diversity, although the differences between all methods are much less prominent. By juxtaposing these observations with the success rates presented in Table 10.1, we conclude that on the categorical problems, relatively low behavioral diversity (around 0.4) is sufficient for achieving high success rates. This observation is confirmed by the results of DOF, which performs the best on these problems (cf. Table 10.1). IFS, despite maintaining the highest diversity, systematically attains lower success rates than DOF.

The figures presented here show that the runs employing discovery of search objectives, either via clustering of factorization, tend to maintain higher levels of diversity than conventional evolutionary search driven by a scalar evaluation. On some problems, DOF and DOC even surpass IFS, which we find remarkable given that IFS was designed to maintain population diversity. Even though DOF was not designed for diversity maintenance, it tends to produce relatively high behavioral diversity, while also finding the most successful candidate solutions.

Table 10.7 provides an in-depth analysis of the dominance relation induced by the best performing DOF variant, i.e DOF-WH with $r = 3$. The first two columns present the percentage of pairs of candidate solutions in S such that one solution dominates the other in G ($\succ_G$) and G' ($\succ_{G'}$), respectively. The next column characterizes the frequency of committing false positive errors (FP). We report it as the percentage of dominances introduced in G' that are absent in G. In the fourth column (INV), we report the percentage of dominance inversion in G'. The last column (PRE) quantifies DOF's capability to preserve the dominance in G' as the percentage of cases in which the relation is consistent in both G and G'. Notice that $1 - PR$ is equal to the amount of false negative errors, and for this reason we do not report it as a separate column. The results are reported for individual benchmark problems as the averages over 200 evolutionary generations and 50 independent runs, accompanied by 95% confidence intervals.

The amount of dominance in G clearly suggests that very few candidate solutions dominate one another. To be precise, the percentage of dominance never exceeds 8.45 percent and its mean value across all problems, reported in the bottom row of Table 10.7, is 3.08 percent. Not surprisingly, the percentage of dominance decreases as the number of tests increases; see for example Cmp6 and Cmp8. However, the distribution of values across all problems also suggests that it also decreases

Table 10.7: Analysis of the dominance relation induced by DOF. The table reports the percentage of dominances in G ($\succ_G$) and G' ($\succ_{G'}$), the frequency of committing false positive errors (FP), the frequency of inversions of dominance (INV), and the percentage of preserved dominances in G' (PRE).

Problem	$\succ_G$		$\succ_{G'}$		FP		INV		PRE	
Cmp6	2.96	±0.09	4.31	±0.22	3.71	±0.23	0.00	±0.00	24.06	±2.22
Cmp8	1.19	±0.13	3.90	±0.24	3.70	±0.24	0.00	±0.00	20.44	±4.05
Maj6	3.59	±0.21	8.12	±0.31	6.80	±0.40	0.00	±0.00	43.61	±6.13
Maj8	1.16	±0.12	6.83	±0.31	6.49	±0.33	0.00	±0.00	34.86	±3.67
Mux6	2.22	±0.12	2.61	±0.22	2.25	±0.16	0.00	±0.00	18.77	±3.40
Par5	0.59	±0.08	0.14	±0.05	0.03	±0.00	0.00	±0.00	18.63	±7.40
Dsc1	0.61	±0.08	18.65	±1.22	18.22	±1.24	0.00	±0.00	88.30	±2.17
Dsc2	0.95	±0.09	22.90	±1.38	22.25	±1.41	0.00	±0.00	91.01	±1.94
Dsc3	1.68	±0.12	22.31	±0.99	21.15	±1.01	0.00	±0.00	89.39	±1.39
Dsc4	0.89	±0.07	21.06	±1.51	20.44	±1.51	0.00	±0.00	90.13	±2.67
Dsc5	0.52	±0.05	21.51	±1.45	21.13	±1.47	0.00	±0.00	92.94	±2.34
Mal1	5.79	±0.16	19.37	±1.37	16.05	±1.26	0.00	±0.00	73.42	±3.44
Mal2	7.13	±0.20	22.64	±1.30	18.57	±1.14	0.00	±0.00	75.64	±4.11
Mal3	5.69	±0.19	15.15	±1.00	11.84	±0.77	0.00	±0.00	70.04	±5.51
Mal4	8.44	±0.26	22.24	±1.19	17.29	±0.98	0.00	±0.00	75.89	±3.58
Mal5	5.79	±0.20	21.62	±1.46	18.07	±1.22	0.00	±0.00	79.20	±4.84
Mean:	3.080		14.591		13.005		0.000		61.651	

with the relative difficulty of problem, measured as the number of runs that failed to find a perfect candidate solution. For instance, the percentage of dominance for Par5 and Dsc4, which are the most difficult problems in our benchmark suite (cf. low success rates in Table 10.1), are 0.59 and 0.897, respectively; these are among the lowest values reported in Table 10.7. These results empirically confirm our intuition that the dominance relation in G is rather sparse.

Given these observations, it is arguably interesting to ask whether DOF manages to maintain higher level of dominance in G'. The results in the second column of Table 10.7 answer this question. Clearly, the compression of G into a few derived search objectives in G' leads to a noticeable increase in the likelihood of dominance in G'; on average, it amounts to 14.59 percent. For individual benchmarks, the increase may be however much more significant. For instance, in case of Dsc2 it is well beyond 21 percent. Notably, DOF maintains higher percentage of dominance for all problems. The most obvious explanation is that by design G' has fewer columns (three in this analysis); low number of search objectives derived by DOF naturally facilitates more dense dominance relation in G'.

The percentage of dominance inversions is zero for all benchmark problems, which implies that DOF never committed this type of error in our experiments. Based on Theorem 10.6, we may expect such a result only when $r >= rank(G)$. In our experiments, we typically have that $r < rank(G)$, which implies that exact NMF is often not possible. The obtained result may however suggest that the approximation error in NMF is sufficiently low so as not to invert the dominance relation. On the other hand, the percentage of FP errors committed by DOF is rather low for Boolean problems and slightly higher for algebra benchmarks, never exceeding 7 and 23 percent, respectively. On average, it amounts to roughly 13 percent. At first sight this result might appear unexpected, particularly in the light of Theorem 10.8. Recall however that this theorem also assumes exact NMF; the observed FP errors are thus caused by heuristic nature of employed NMF algorithm (cf. Section 10.1). Nevertheless, based on Theorem 10.8, we may expect the percentage of FP errors to approach zero as the factorization error (10.1.2) also approaches zero.

On one hand, FP errors may appear undesirable as they clearly distort the original information in G. On the other, the capability to posit a dominance in G' that was factually absent in G may also be considered valuable. Consider for instance a situation in which a candidate solution s_1 solves only easy tests, while a candidate solution s_2 solves a more diversified set of tests, which includes some easy and some difficult tests, and is disjoint with the set of tests solved by s_1. Even though they are incomparable in the sense of $\succ_G$, there is a chance that $s_2 \succ_{G'} s_1$. The latter may be beneficial for the search process, because s_2 can be prospectively a more promising candidate solution. Nevertheless, the experimental results seem to suggest that FP errors do not have any significant impact on success rates; problems with 0 and 100 percent success rates tend to have very similar percentage of FP errors (cf. Cmp6 and Cmp8, Maj6 and Maj8, Dsc3 and Dsc4 in Table 10.1).

Despite not preserving the dominance relation between candidate solutions in general, it is interesting to see that DOF manages to transfer 61.65 percent of these relations to G' on average. For Boolean problems, this percentage ranges from 18.77 to 43.61 percent. For categorical problems, it is noticeably higher and varies between 70.05 and 92.95 percent. The actual impact of the preserved dominances on the selection process is unfortunately hard to assess. Recall that DOC guarantees preserving the dominance relation and tends to achieve worse success rates than DOF; this gives us some premises to assume that this characteristics is of secondary importance to DOF. Another argument for this is that there is typically not much to preserve in the first place, as evidenced by the percentage of dominances present in G (the first column of Table 10.7). On the other hand, the results for Boolean problems seem to suggest that success rates positively correlate with the amount of dominance preserved (which might be however a statistical fluke).

To conclude our considerations, the experimental evidence presented here suggests that DOF maintains higher levels of behavioral diversity, which we hypothesize contributes positively to the observed increases in performance. Although maintaining higher levels of diversity may be overall beneficial, simply maintaining a semantically diverse set of candidate solutions does not single-handedly help find an ideal solution without sufficiently strong pressure toward the goal. This is particularly evident in the case of IFS, which maintains even higher diversity than DOF on some benchmark problems, yet tends to achieve systematically lower success rates. It seems therefore that, in combination with the increased diversity, DOF manages also to provide an adequate search gradient to exploit good candidate solutions, which we believe is enabled by the relatively high count of dominances between candidate solutions in G'. Without it, the dominance relation employed by any multiobjective selection method would not have grounds for deeming one candidate solution better than other, resulting in a weak search gradient. The evidence gathered in this section suggests thus that DOF's capability to maintain high levels of diversity while also applying diversified search gradient is largely responsible for its success on the benchmark problems presented here.

10.5.5 Visualization of search objectives

In the previous sections, we focused on the impact of derived search objectives on various aspects of search performance, neither investigating the relations between them, nor paying much attention to how candidate solutions exploit the trade-offs between them. To provide a deeper insight into this aspect of search objectives, Fig. 10.3 visualizes the objectives derived by DOF in the last generation of a single randomly selected run for each of our benchmark problems. For the purpose of creating the graphs, we employed DOF-WH with factorization rank $r = 2$, so that exactly two search objectives are derived in each generation of evolutionary run. For each row in G', a blue

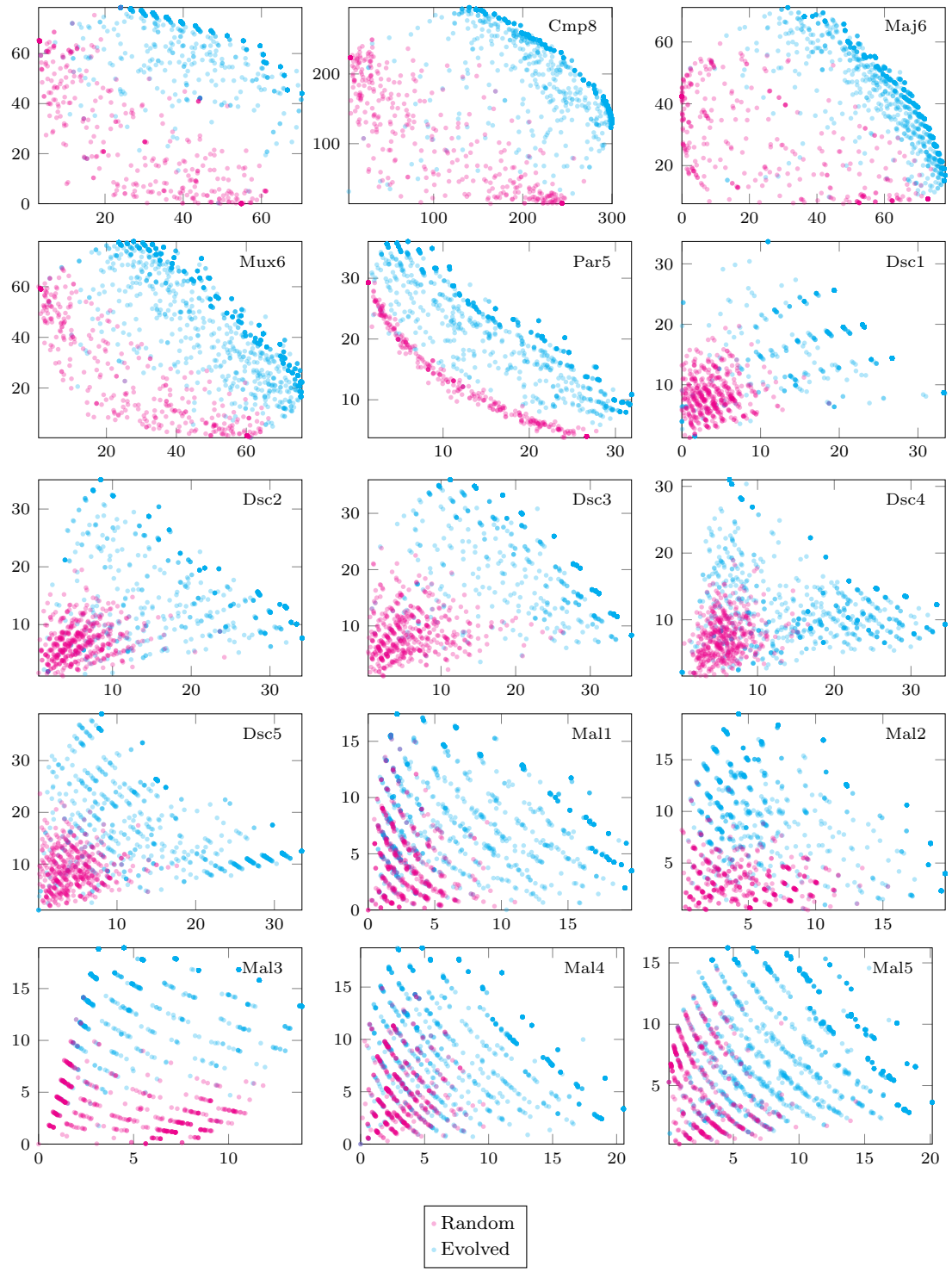


Figure 10.3: Visualization of search objectives derived by DOF-WH with $r = 2$. The axes correspond to two derived objectives, and each point reflects the candidate solution's performance on these objectives. The evolved candidate solutions are marked in blue, the random ones in magenta. Color saturation reflects their density.

Table 10.8: Pearson's correlation coefficient averaged over all pairs of search objectives and across 50 evolutionary runs. Each column corresponds to a different generation of evolutionary search.

Problem	1	10	25	50	100	150	200	Mean
Cmp6	−0.626	−0.648	−0.665	−0.625	−0.642	−0.689	−0.683	−0.654
Cmp8	−0.774	−0.739	−0.716	−0.680	−0.794	−0.627	−0.713	−0.720
Maj6	−0.536	−0.545	−0.552	−0.553	−0.541	−0.509	−0.509	−0.535
Maj8	−0.829	−0.771	−0.789	−0.786	−0.662	−0.726	−0.652	−0.745
Mux6	−0.672	−0.639	−0.620	−0.558	−0.558	−0.558	−0.558	−0.595
Par5	−0.847	−0.864	−0.832	−0.812	−0.779	−0.810	−0.747	−0.813
Dsc1	−0.199	−0.510	−0.462	−0.407	−0.409	−0.481	−0.420	−0.413
Dsc2	−0.172	−0.660	−0.587	−0.602	−0.614	−0.612	−0.615	−0.552
Dsc3	−0.530	−0.733	−0.710	−0.735	−0.612	−0.757	−0.758	−0.691
Dsc4	−0.093	−0.407	−0.358	−0.345	−0.365	−0.371	−0.390	−0.333
Dsc5	−0.100	−0.700	−0.559	−0.587	−0.475	−0.408	−0.275	−0.443
Mal1	−0.227	−0.369	−0.381	−0.373	−0.274	−0.274	−0.274	−0.310
Mal2	−0.172	−0.335	−0.332	−0.331	−0.334	−0.331	−0.319	−0.308
Mal3	−0.523	−0.545	−0.502	−0.548	−0.583	−0.584	−0.473	−0.537
Mal4	−0.214	−0.420	−0.467	−0.230	−0.479	−0.447	−0.567	−0.403
Mal5	−0.206	−0.196	−0.168	−0.165	−0.165	−0.165	−0.165	−0.176
Mean:	-0.420	-0.568	-0.544	-0.521	-0.518	-0.522	-0.507	

points marks the performance of a candidate solution on the search objectives. Following our previous visualizations (cf. Section 9.4.9 and Fig. 9.6), we also plot the performance of random candidate solutions. In DOF, discovery of search objectives is a one-off process, i.e. once G is factorized into a product of W and H, it is no longer possible to derive objectives for candidate solutions not present in G, unless the entire process is repeated for a new interaction matrix. To circumvent this limitation, we generate randomly 500 extra candidate solutions that are subject to evaluation on all tests exactly as regular candidate solutions. The outcomes of their interactions with tests are then added to G just before applying NMF, leading to extra rows in W that do not affect the rows corresponding to evolved candidate solutions in any significant way. Let us note that these actions do not impact search in any meaningful way, and are only performed for the purpose of preparing the graphs. The points obtained in this way are plotted in magenta on each graph. Where the marks overlap, color saturation reflects their density.

Each inset in Fig. 10.3 corresponds to a different pair of search objectives, specific to a problem being solved, a run, and the population of candidate solutions at the end of run. Not surprisingly, the evolved candidate solutions follow a similar trend as in Fig. 10.3, spanning almost uniformly the space between the axes of objectives. The best candidate solutions clearly approximate the Pareto-front, exploiting the trade-off between objectives in various ways. The random solutions, on the other hand, are typically clustered in the bottom left corner of the graphs, which implies low performance on both objectives. In fact, they are dominated by the vast majority of evolved candidate solutions, hardly ever drawing anywhere near the Pareto-front of evolved candidate solutions. Interestingly, the evaluation provided by the derived search objectives appears to be very meticulous; even random solutions are partitioned into multiple Pareto ranks (though it is also possible that those layers simply result from the discrete nature of interaction outcomes). This is particularly evident on graphs that belong to algebra benchmarks (e.g. Mal5). The individual graphs provide evidence that DOF maintains a high level of diversity even at the end of evolution.

The graphs also suggest that the search objectives derived by DOF are strongly decorrelated. To provide some evidence that this is indeed true, in Table 10.8, we report the value of Pearson's correlation coefficient in several generations of evolutionary search, averaged over all pairs of ob-

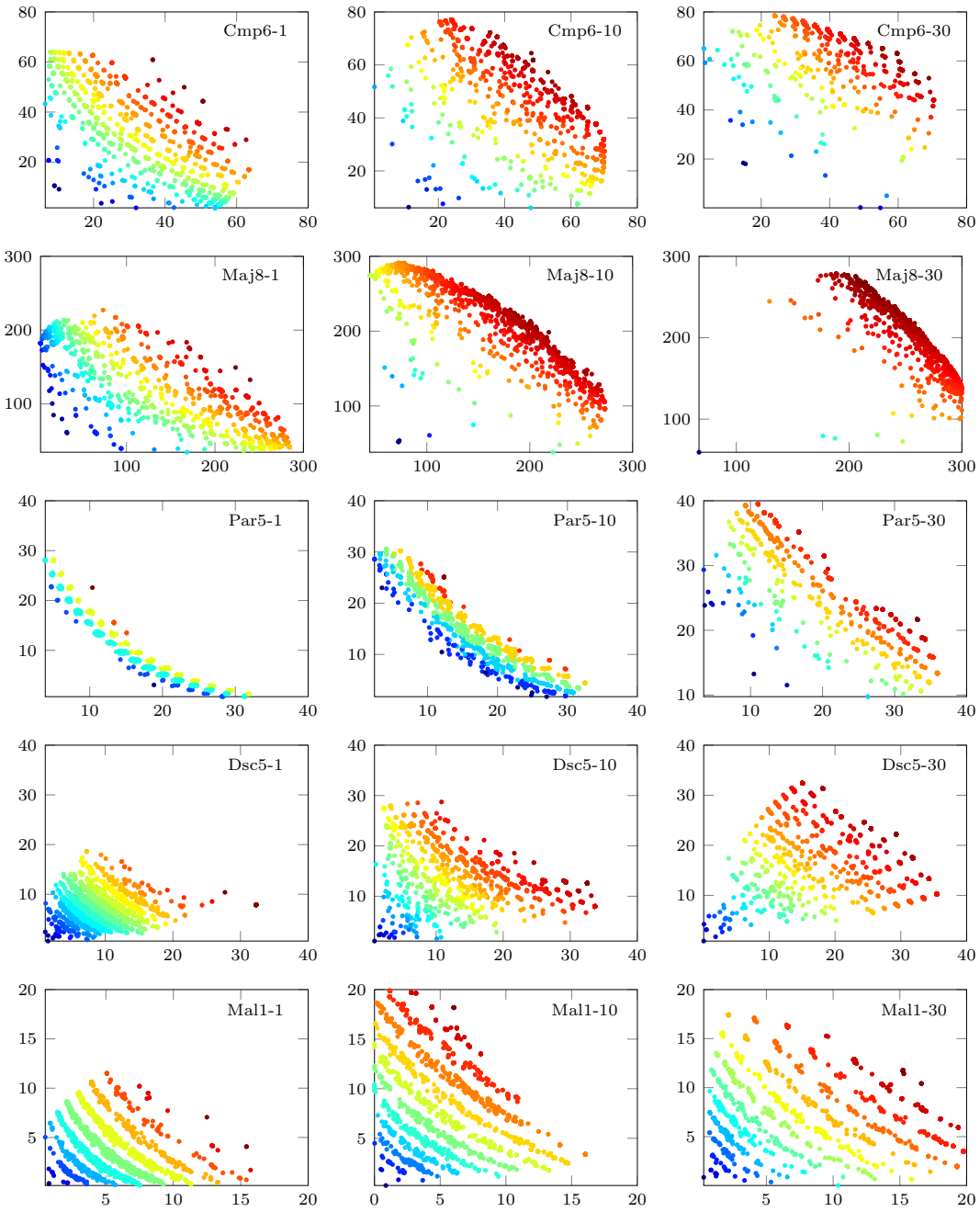


Figure 10.4: The relation between search objectives derived by DOF-WH with $r = 2$ and the number of solved tests (fitness), plotted for selected benchmark problems (in rows) at three different points of evolution (in columns). The numbers next to benchmark names indicate the generation number. Each points corresponds to a single candidate solution and reflects its performance on two search objectives. The color denotes their fitness, ranging from dark blue for low values to dark red for high values of fitness.

jectives and across 50 runs (technically, these are thus average correlations between all pairs of columns in W). The results clearly demonstrate that the average correlation between objectives is always negative. In other words, an increase in performance on one objective is associated with a decrease on the other. This strongly suggests that individual search objectives measure different aspects of candidate solution's quality that are in conflict with each other. Judging from the averages across the benchmarks problems reported in the bottom row of Table 10.8, the objectives derived in the early stages of evolution are characterized by a slightly weaker negative correlation than those derived later. Furthermore, the observed decorrelation appears to be stronger for the difficult problems. For instance, the average correlation coefficient across all generations, reported in the last column of Table 10.8, is the lowest for Par5 and Maj8, which, as the previous results have shown, are arguably the hardest problems in our benchmark suite. This might suggest that it is easier to derive search objectives for problems that are inherently more difficult. We also anticipate such problems to feature more than two underlying objectives, hence it might be beneficial to let DOF work with more objectives in such cases.

Search objectives are intended to provide alternative multi-aspect characterization of candidate solutions on one hand, and to make search algorithms capable of exploiting that characterization on the other. Concerning the former, it becomes justified to ask if candidate solutions that score high in the eyes of conventional evaluation function are also valuable according to search objectives, and vice-versa. In an attempt to answer this question, in Fig. 10.4, we plot candidate solutions in such a way that their coordinates correspond to performance on particular search objectives, while their color reflects fitness meant as the number of solved tests (the colors range from dark blue for low values of fitness, through yellow in the middle, to dark red at the high end). In the rows of Fig. 10.4, we presents these graphs for five selected problems (Cmp6, Maj8, Par5, Dsc5, Mal1). To investigate how temporal changes in a population of candidate solutions affect the relation between fitness and search objectives, each column of Fig. 10.4 is associated with a different point in evolution, staring from the very first generation of search (the leftmost column), after ten generations have elapsed (the middle column), and after one hundred generations (the rightmost column).

The common feature of all graphs is that the search objectives corresponding to both axes are positively correlated with fitness, i.e. the higher a candidate solution scores on any objective, the more tests it solves. The figure also suggests that scalar fitness fails to differentiate many candidate solutions that actually behave very differently in terms of performance on the derived search objectives. This trend appears to continue over the course of evolution, as suggested by the subsequent graphs in each row. The figure reveals also how candidate solutions gradually move in the direction of upper right corner, while maintaining overall high diversity, in the process. Notice also how in some cases high fitness corresponds to a rather small portion of Pareto-front, which means that many promising candidate solution may be discarded prematurely when search is driven by scalar fitness function.

These observations together suggest that the search dynamics of DOF is significantly different from that of GP. However, the empirical results presented in this chapter provide strong evidence that DOF is more advantageous when it comes to the probability of finding an optimal solution.

10.5.6 Computational overhead

The process of discovering search objectives via factorization incurs additional computational cost which in DOF originates from two sources: performing the NMF (using the MU algorithm) and carrying out multiobjective selection using the NSGA-II. The computational complexity of the MU

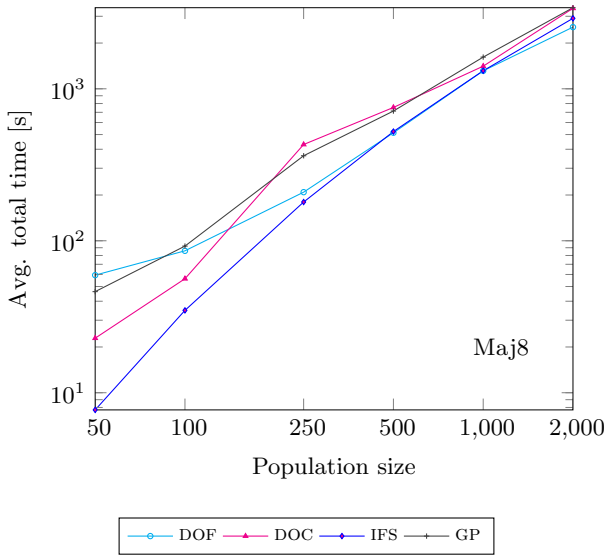


Figure 10.5: Scaling of wall-clock runtimes as a function of population size.

algorithm discussed in Section 10.1 is of the order of $O(mnr)$ per iteration. Taking into account the complexity of multiobjective selection via NSGA-II algorithm and assuming i iterations of NMF, the total overhead is $O(mnri+nm^2)$, which is actually similar to the overhead of DOC (cf. Section 9.6), given that r is typically very small and NMF may terminate earlier than after the assumed 50 iterations when the reconstruction error drops below the assumed threshold. Given that, we expect the extra costs of NMF to become only a small fraction of the total cost of evaluation in many test-based problems.

In order to get an empirical sense of how DOF scales with respect to some parameters of problem instance in comparison to the control methods, we run a series of experiments on one of the hardest problem in our benchmark suite, i.e. Maj8 (cf. Table 10.1). We vary the population size m between 50 and 2000, and let each method search for an ideal solution for 200 generations. We then select 10 runs for each configuration of method and population size that failed to find an ideal candidate solution. This allows us to ignore any runs that ended earlier, making it safe to assume that each configuration is given the same computational budget. We use the results of these experiments to estimate the time complexity of DOF as a function of m. The computations were conducted on a cluster of uniform PCs in Poznań Supercomputing and Networking Center, with 2.6 GHz Intel Xeon E5-2697 processors and 64 GB of memory.

The results of the time complexity experiment are shown in Fig. 10.5 as a log-log plot, with average wall-clock times on the y-axis and the population size on the x-axis. We estimate the runtime t as a function of population size m by finding a linear trend via least-squares that is best-fit to the log transformed data, i.e.:

$$\log(t) = a\log(m) + b, \tag{10.5.2}$$

A couple of straightforward transformations give us the time scaling formula as $t = cm^a$, where $c = \exp(b)$. For DOF, $a = 1.07$ and $b = -0.328$ (the determination coefficient for the regression model is $r^2 = 0.987$). Therefore the runtime of DOF as a function of m is estimated to $t = 0.72m^{1.07}$, which suggests the time complexity with respect to m to be much more optimistic than DOF's worst case complexity of $O(mnri + nm^2)$. In general, DOF falls between GP and IFS in terms of wall-clock times (cf. Fig. 10.5). At higher population sizes ($m \geq 500$), it achieves the lowest run times, on par with IFS. DOF's tendency to evolve smaller programs (cf. Section 10.5.3) is

Table 10.9: Average evaluation and factorization time in DOF, accompanied by 95% confidence intervals. The last column reports the percentage of evaluation time taken by NMF.

Problem	Eval [s]		NMF [s]		Ratio [%]
Cmp6	1.262	±0.024	0.113	±0.001	8.93
Cmp8	1.671	±0.043	0.307	±0.002	18.39
Maj6	1.393	±0.026	0.117	±0.001	8.38
Maj8	2.867	±0.073	0.321	±0.005	11.21
Mux6	0.727	±0.010	0.114	±0.001	15.68
Par5	2.505	±0.060	0.097	±0.000	3.87
Dsc1	0.945	±0.016	0.095	±0.000	10.09
Dsc2	0.956	±0.015	0.095	±0.000	9.94
Dsc3	1.078	±0.017	0.095	±0.000	8.81
Dsc4	0.982	±0.018	0.095	±0.000	9.64
Dsc5	0.761	±0.017	0.095	±0.000	12.48
Mal1	0.639	±0.009	0.087	±0.001	13.56
Mal2	0.634	±0.012	0.089	±0.001	14.09
Mal3	0.881	±0.015	0.086	±0.001	9.76
Mal4	0.947	±0.019	0.083	±0.001	8.76
Mal5	0.414	±0.003	0.087	±0.001	21.01
Mean:	1.166		0.124		11.54

surely helpful here, as smaller programs take less time to evaluate. DOC, on the other hand, is on average more computationally expensive (complexity model $t = 0.13m^{1.36}$), particularly for larger population sizes.

It is also interesting to investigate the impact of NMF on evaluation time, i.e. how much the time needed to evaluate candidate solutions is lengthened by NMF. To this end, in Table 10.9, we report the average total evaluation time (the time needed to conduct the interactions between candidate solutions and tests, and to derive search objectives) and factorization time, accompanied by 95% confidence intervals for particular methods and benchmarks considered in this chapter. Since evaluation subsumes discovery of search objectives, and by this token also the process of NMF, in the last column of Table 10.9, we report the percentage of evaluation time taken by NMF. The empirical evidence gathered from our experiments suggests that the overhead of NMF compared to the overall evaluation time is small, 11.54 percent on average, and never exceeds 21.01 percent. It is also worth pointing out that these numbers could be further reduced by using more efficient implementation of NMF (possibly even GPU-based) or, e.g., limiting the number of iterations used by NMF algorithm. We expect that sufficiently good factorization for our needs can be found after only a few iterations, since the *precision* of reconstruction of the interaction matrix G is of secondary importance – it is the order of candidate solutions induced by W and H that matters, which we expect not to change too much after the very first iterations.

10.6 Discussion

Of the two proposed variants, DOF-W and DOF-WH, the latter performs consistently better on our benchmark suite (cf. Section 10.5.2). The most obvious explanation is that DOF-W, by relying merely on W to derive objectives, uses only part of the information elicited by NMF. In certain scenarios, cursory feedback offered by W may even turn deceptive, as demonstrated by decreasing performance of DOF-W for higher values of factorization rank r. In contrast, DOF-WH employs both matrices, W and H, which allows it to embrace the entirety of rich information learned during factorization to derive better informed search objectives that lead to systematic improvements in

performance. Let us also note the DOF-WH presented here is an enhanced version of the original DOF-WH variant proposed in [213].

The use of NMF in DOF is motivated by its capability to *explain* the interaction outcomes in G, i.e., characterizing both candidate solutions and tests in terms of the latent factors inferred from the patterns observed in their interactions. The latent factors may for instance measure the degree to which a candidate solution possess certain skills necessary to pass individual tests and reflect how well it deals with tests that require that skill. Factorizing an interaction matrix allows us thus to discover the most descriptive dimensions for characterizing candidate solution's capabilities. It is however hard to provide unambiguous interpretation of the factors learned by NMF, especially in the context of GP, where the mapping from program syntax to semantics is very complex. One interesting possibility would be to identify the first few most important dimensions from a matrix decomposition and juxtapose the candidate solutions' location in this space with, e.g., its syntactic representation. We would not be surprised to find out that NMF groups together candidate solutions that maintain semantically similar pieces of code.

We decided to keep r constant during evolution, because automatic tuning of r in NMF is rather difficult, mainly due to the prohibitively large computation costs associated with such a selection. For instance, a natural choice in many applications is to perform a cross-validation and pick r that leads to the lowest approximation error. Cross-validation requires however computing multiple independent factorizations, which is too expensive to perform in every generation of an evolutionary run. Another interesting possibility involves tracking the decay of singular values of the interaction matrix. This approach essentially boils down to computing singular value decomposition, which is not cheap either. Nevertheless, given overall good performance of DOF for fixed r, we believe that any potential performance gain resulting from automatic selection of r may not be worth extra computational effort.

A vigilant reader has surely noticed the absence of experiments with CoEAs in this chapter. This decision is mainly motivated by the overwhelmingly positive performance of DOF on GP problems, giving us a strong reason to believe that good performance of DOF is not incidental, and would likely carry over to CoEAs. Another argument that justifies this claim is that both DOF and DOC are essentially agnostic about the actual metaheuristic powering the search. They only care about an interaction matrix, which makes them viable methods for any test-based problem.

One may claim that neither DOF nor DOC ultimately solve the evaluation bottleneck problem (cf. Section 6.1), since in place of a single scalar objective to drive a search, we derive up to four objectives, which may be considered very little compared to the abundance of information available in an interaction matrix. On the other hand, using a small number of objectives is likely to cause the dominance relation in the derived space to be dense enough to provide a necessary search gradient. This observation is corroborated by the experimental evidence presented in this chapter, which shows that even 'modest' widening of evaluation bottleneck significantly boosts the likelihood of finding an ideal candidate solution, compared to conventional single-objective search techniques.

The formalism of matrix factorization employed in this chapter creates also other interesting opportunities that are worth mentioning here. In particular, NMF can be computed even for sparse matrices. This makes it possible to perform interactions only between some pairs of candidate solutions and tests, place their outcomes in G, and let the NMF generalize this data in a way that allows predicting unknown outcomes of interactions. In Chapter 12, we further build on this idea, giving rise to a range of approaches aimed at reducing computational cost incurred by evaluation. Furthermore, NMF applied to an interaction matrix facilitates finding semantically

similar candidate solutions and/or tests of similar difficulty, which opens the door to pursuing, e.g., a family of recommendation-driven search operators.

10.7 Chapter summary

In this chapter, we proposed a novel a algorithm, DOF, that combines the technique of non-negative matrix factorization with the metaheuristic of evolutionary algorithms. DOF replaces the original scalar evaluation function with heuristic search objectives defined using the latent factor space that results from applying NMF to an interaction matrix. By doing so, it clearly subscribes to the proposed framework of discovery of search objectives (Chapter 8).

By scrutinizing individual interaction outcomes, DOF goes even further than DOC in harvesting information about candidate solutions and their behavior. Crucially, this enables DOF to derive a multi-faceted characterization of candidate solutions by means of search objectives that systematically leads to higher success rates in comparison to the conventional EAs driven by the scalar evaluation (Section 10.5.2). Test-based problems that proved very hard to solve for conventional single-objective search often become tractable when approached by DOF.

The experiments reported in this chapter demonstrate that DOF is able to derive meaningful search objectives (Section 10.5.5) that lead to an increased behavioral diversity over the course of evolution and a strong search gradient that facilitates discovery of high-quality candidate solutions (Section 10.5.4). The empirical evidence gathered from our experiments suggests also small overhead of the method (Section 10.5.6), encouraging application of DOF in practice.

Chapter 11

Discovery of Search Objectives in Continuous Domains

Empirical evidence brought in the previous two chapters suggests significant practical utility of the framework for discovery of search objectives in test-based problems. The two concrete implementations of the framework, DOC and DOF, not only address the problem of evaluation bottleneck, but also prove more effective than conventional (co)evolutionary search that navigates the search space using scalar fitness function. These methods are however limited in being applicable only to problems with constrained interaction outcomes, which includes binary domains where candidate solutions either pass a test or not (exemplified with the benchmarks used in the previous two chapters), and domains where interaction outcomes are continuous (or at least ordered) but constrained (e.g., games with multiple yet constrained payoff function). Given that a significant fraction of test-based problems involves an interaction function with unconstrained continuous outcomes (e.g., symbolic regression, see Section 5.4.5), it becomes natural to seek for means of extending such approaches in that direction. For instance, in symbolic regression, which is of our primary interest in this chapter, outcomes of interactions could reflect absolute errors made by candidate solutions on particular tests (cf. Sections 4.4 and 8.2)

The contribution of this chapter is thus a largely universal approach for transforming interaction matrices generated in continuous domains (Section 11.1) to a form that is appropriate for DOC, DOF and their variants (those presented in this thesis and the potential ones). Apart from presenting and motivating this particular method, we thoroughly assess its performance on a range of uni- and multivariate symbolic regression benchmarks (Section 11.2). Additionally, we hybridize our approach with lexicase selection and carefully evaluate the resulting configurations.

The method presented in this chapter has been originally published [216].

11.1 Mapping continuous errors to interaction outcomes

DOC and DOF have been originally designed with binary interaction outcomes in mind. Technically, we expect them to still perform well for continuous or multi-valued ordered interaction outcomes, because the algorithms they are built upon behave well in such scenarios: the k-means algorithm in DOC can handle arbitrary distributions of data points, and the MU algorithm in DOF operates on arbitrary non-negative values. However, in order to apply them to test-based problems with unconstrained interaction outcomes, we introduce a preprocessing step that maps the – in general arbitrary large – continuous errors to interaction outcomes. It is however important to emphasize that by 'unconstrained' we mean here *real-valued non-negative* interaction

outcomes. In practice, they will typically have the interpretation of errors, e.g. approximation errors in symbolic regression. i.e. $g_{ij} \in \mathbb{R}_{\geq 0} \cup \{\text{NaN}\}$, where NaNs signal execution errors (like division by zero etc.). For regression tasks considered in this chapter, g_{ij} is the absolute deviation of candidate solution's output w.r.t. desired output y_j, i.e. $g_{ij} = |s_i(x_j) - y_j|$ (cf. Section 8.2). Notice that this reverses the convention adopted in the previous chapters, where greater values of interaction outcomes were more desirable.

The preprocessing method proceeds as follows:

1. Calculate the interaction matrix G between the candidate solutions from the current population S and the tests from T.
2. Replace in G the NaNs and the values that are greater than a threshold φ with the positive infinity $+\infty$.
3. Standardize the columns of G, omitting any infinities.
4. Apply sigmoid 'squeezing' non-linearity $1/(1 + e^{-x})$ to the elements of G.

The last step maps the unconstrained range of original interaction outcomes onto the closed interval $[0, 1]$ and also aims at reducing the impact of outliers on clustering in DOC and factorization in DOF. Such outliers may arise when candidate solution's output $s_i(x_j)$ diverges greatly from the desired output y_j, and propagate through the earlier steps of the above procedure (notice that they are only linearly scaled in step 3). If not handled in Step 4, they could make it difficult to detect meaningful cluster centers in DOC and arrive at good factorizations in DOF.

As a result of applying the above preprocessing steps, the values in G are now guaranteed to be in the range $[0, 1]$. Crucially, the smaller φ, the more sensitive the method becomes to small errors. For example, assuming $\varphi = 200$ and the following errors made by a single individual on six test cases in the initial G:

$$0.3 \quad 1.6 \quad 3.3 \quad 150.8 \quad 51.3 \quad 160.3$$

none of them would be omitted during standardization. Moreover, relatively small interaction outcomes for the second and third elements become indiscernible after standardization and squeezing:

$$0.29 \quad 0.3 \quad 0.3 \quad 0.79 \quad 0.46 \quad 0.81,$$

which may have consequences for further processing, in particular for selection. If, on the other hand, we set $\varphi = 50$, only the first three errors would be considered for computing mean and standard deviation necessary for standardization. As a result, the differences between the smaller errors are amplified:

$$0.24 \quad 0.47 \quad 0.78 \quad 1.0 \quad 1.0 \quad 1.0.$$

The motivation for parameterizing this process with φ is that maintaining the differences between smaller errors may be more important than for the larger errors, as for the latter it may be justified to consider them 'equally bad'. In the experimental section, we propose to automate the choice of φ by setting it to the 95th percentile of the errors in an interaction matrix G, which proves effective in empirical evaluation.

Other design choices such as standardization and sigmoid non-linearity are motivated by the importance of providing for the same magnitude of outcomes on tests, while maintaining their individual capability of discrimination between candidate solutions. The former is particularly important for DOC, as it employs clustering based on Euclidean distances that is isotropic of space and therefore sensitive to unequal variances of observations in different dimensions (dimensions that have larger attributes have greater impact on the distance).

11.2 Experiments

We conduct an extended experimental assessment of DOC and DOF by confronting them with reference approaches on a suite of 18 symbolic regression benchmarks. The primary objective of the experiment is to gauge the performance of these methods in terms of typical metrics such as training set error, test set error and program size, in the domain of tree-based GP.

To this point, we employed search objectives in a purely multi-objective fashion; with this experiment we intend to find out whether search objectives lend themselves to a different selection techniques. With this in mind, our secondary objective is to assess usefulness of lexicase selection (cf. Section 8.5.3) when applied to derived search objectives. We also intend to gain some deeper insight into the differences between the methods by scrutinizing their behavior on our benchmark problems.

11.2.1 Compared algorithms

The experimental setup is exactly the same as in our previous experiments (cf. Table 9.3). Following common practice, a program is deemed correct when $f_T(s) < 2^{-23}$, where f_T (5.3.2) is program's square error on the set of tests T.

DOC and DOF are implemented as described in Section 9.1 and 10.2, respectively. The only difference lies in how an interaction matrix is processed prior to derivation of search objectives. To account for arbitrary large continuous interaction outcomes (errors), we employ the mapping approach described in Section 11.1. As already signaled there, we set φ to 95th percentile of the errors in an interaction matrix. This allows us to ignore the top 5 percent of the largest errors that would otherwise distort the standardization of G. In the preliminary experiments, we also experimented with median, which turned out to perform slightly worse on average. As a result of the preprocessing step, outcomes of interactions in G are guaranteed to be in the range $[0, 1]$, enabling both DOC and DOF to proceed as originally designed (where, however, the interaction outcomes in DOF are technically incremented by 1, to distance them from 0; see Section 10.2). We consider those configurations of DOC and DOF that performed the best in the experiments conducted in Chapters 9 and 10, i.e., DOC with X-MEANS ($k = [1, 5]$) and DOF-WH ($r = 3$).

We confront the above methods with several control setups. The first of them is the conventional Koza-style GP (cf. Section 9.5.1). The other control methods are shortly introduced in the following subsections.

ϵ-lexicase selection

ϵ-lexicase selection (LEX) has been recently proposed for symbolic regression problems [197], and builds upon lexicase selection that has been originally designed for 'uncompromising' problems [127]. ϵ-lexicase selection addresses poor performance of lexicase selection on continuous errors by modulating the pass condition c_t (cf. Section 8.5.3) that controls the subset of tests that are allowed to pass to the next iteration of selection process. In the following, we use the variant of ϵ-lexicase selection that proved to be the most effective in [197] and uses pass condition defined as:

$$e_t(s) < e_t^* + \lambda(\mathbf{e}_t), \tag{11.2.1}$$

where $e_t(s)$ is the error of candidate solution (program) s on test t, e_t^* is the smallest error committed on t by the programs in the current population S, and $\lambda(\mathbf{e}_t)$ is the median absolute deviation of the errors on test t across the population.

Similarly to DOC, in addition to rewarding the programs for solving test cases, LEX promotes diversified programs that pass randomly selected subset of tests. In this way, different tests are emphasized in each selection event. An individual that passes test(s) that are rarely passed by its competitors has substantial chance to propagate to the next generation even if it performs poorly on many other tests. Note that in contrast to DOC and DOF, LEX does not explicitly define any objectives or alternative fitness functions. In this sense, it is 'natively' a selection method.

Age-Fitness Pareto Optimization

Age-Fitness Pareto Optimization (AFPO) [316] is a multi-objective method that assigns each individual an *age* equal to the number of generations it has been present in the population. In each generation, AFPO selects random parents from the population and applies crossover and mutation operators to produce $|S| - 1$ offspring. The offspring and a single randomly initialized individual are then added to the population, doubling so its size. Next, Pareto tournament selection on two objectives, fitness (maximized) and age (also maximized), is iteratively applied by randomly selecting a subset of individuals and removing the dominated ones until the size of the population is reduced back to $|S|$.

Hybrid Approaches

In their original form, derived objectives identified by DOC and DOF drive the selection process in a multi-objective fashion to avoid aggregation of interaction outcomes with all tests into a single scalar value, which is characteristic for the traditional evaluation function. One of the main motivations for LEX was also to avoid such aggregation, and the decisions made by the algorithm regarding which programs to select are based on distinct tests (cf. Section 8.5.3). Moreover, compared to NSGA-II used for selection in DOC and DOF, lexicase is naturally prepared to handle larger numbers of objectives. These observations encourage us to combine these methods into hybrid approaches, in which lexicase selection is performed on the derived objectives.

In hybrid approaches, DOCLEX and DOFLEX, we first derive new search objectives and subsequently, we apply LEX using the particular derived objectives as if they were regular test cases. To be precise, the method proceeds as follows. In each iteration, a derived objective is drawn at random. Then, individuals in the population that do not satisfy the pass condition c_t on that objective are filtered. If more than one individual remains, the process repeats, filtering any remaining individuals that do not satisfy c_t on the next derived objective drawn at random. This process continues until only one individual remains and is selected, or until all derived objectives have been processed, in which case a random program is selected from the remaining programs. Based on our experience, LEX tends to work best if there are at least several dozens of tests. For this reason, in DOCLEX, we set $k \in [10, 100]$, and let X-MEANS pick the optimal value of k during clustering. In DOFLEX, r cannot be greater than $\min(m, n)$, so we set $r = n$ in order to derive as many search objectives as there are tests; the motivation for this setting will be brought in the discussion that follows the experiment.

11.2.2 Benchmark problems

We compare the methods on 18 uni- and bi-variate symbolic regression problems, listed in Table 11.1. In univariate problems, 20 Chebyshev nodes [39] constitute the set of tests T, while 20 uniformly sampled points are used for testing. Chebyshev nodes are commonly used as values for independent variables because they protect from Runge's phenomenon [308] (see also [38], Ch

Table 11.1: The symbolic regression task used in the experiment.

Problem	Definition	Variables	Range	$\|T\|$
R1	$(x+1)^3/(x^2-x+1)$	1	$[-1,1]$	20
R2	$(x^5-3x^3+1)/(x^2+1)$	1	$[-1,1]$	20
R3	$(x^6+x^5)/(x^4+x^3+x^2+x+1)$	1	$[-1,1]$	20
Kj1	$0.3x\sin(2\pi x)$	1	$[-1,1]$	20
Kj4	$x^3e^{-x}\cos(x)\sin(x)(\sin(x)^2\cos(x)-1)$	1	$[0,10]$	20
Kj8	$\sqrt{x}$	1	$[0,100]$	20
Kj14	$\arcsin h(x)$	1	$[-3,3]$	20
Kj15	$x^3/5.0+y^3/2.0-y-x$	2	$[-3,3]$	20
Ng3	$x^5+x^4+x^3+x^2+x$	1	$[-1,1]$	20
Ng4	$x^6+x^5+x^4+x^3+x^2+x$	1	$[-1,1]$	20
Ng5	$\sin(x^2)\cos(x)-1.0$	1	$[-1,1]$	20
Ng6	$\sin(x)+\sin(x^2+x)$	1	$[-1,1]$	20
Ng7	$\log(x+1)+\log(x^2+1.0)$	1	$[0,2]$	20
Ng8	$\sqrt{x}$	1	$[0,4]$	20
Ng9	$\sin(x)+\sin(y^2)$	2	$[0,1]^2$	100
Ng12	$x^4-x^3+(y^2/2)-y$	2	$[0,1]^2$	100
Pg1	$1/(1+x^{-4})+1/(1+y^{-4})$	2	$[-5,5]^2$	100
Vl1	$e^{-(x-1)^2}/(1.2+(y-2.5)^2)$	2	$[0,6]^2$	100

8). For bivariate problems, 10 values are picked in analogous way for each input variable and their Cartesian product defines the set of input variables used in T. The table also presents the mathematical expressions to be synthesized by GP in each problem, the range of values from which the points are sampled, and the cardinality of T. We use the set of instructions that is typical for symbolic regression, i.e $+, -, \times, /, \exp, \log, \sin, \cos$. We use the protected logarithm implemented as $\log|x|$ and the protected division, which returns 0 for a divisor equal to zero [336]. By varying in the number of variables, the required functions, and the characteristic of desired output, this selection of benchmarks forms good representation of problems considered in research on GP and its practical applications [238, 278].

11.2.3 Results

Figure 11.1 shows the average best-of-generation fitness (i.e., error on the training set) achieved by particular methods on different benchmark problems, with 95% confidence intervals marked as semi-transparent bands. Clearly, each of the considered methods that drive the search using derived objectives significantly improves the performance of the standard GP algorithm. The best performance is achieved either by DOC or DOCLEX, depending on the problem. LEX and DOCLEX tend to maintain the lowest training set error during evolution, with DOC eventually catching up. DOF performs slightly worse than the already mentioned methods, but still better than two other control methods AFPO and GP. DOFLEX, on the other hand, seems to be the weakest algorithm of the analyzed ones. It has the highest variance, and often achieves inferior results when compared to other methods. These observations are confirmed by Table 11.2 that shows 5h3 average and 95 confidence intervals of the final best-of-run fitness.

To provide an aggregated perspective on performance, we employ the Friedman's test for multiple achievements of multiple subjects [159] on the best-of-run fitness. The p-value for Friedman test is 3.46×10^{-12}, which strongly indicates that at least one method performs significantly different from the remaining ones. To determine the significantly different pairs, we conduct a post-hoc analysis using the symmetry test [135]. The left inset in Table 11.4 presents the p-values for the

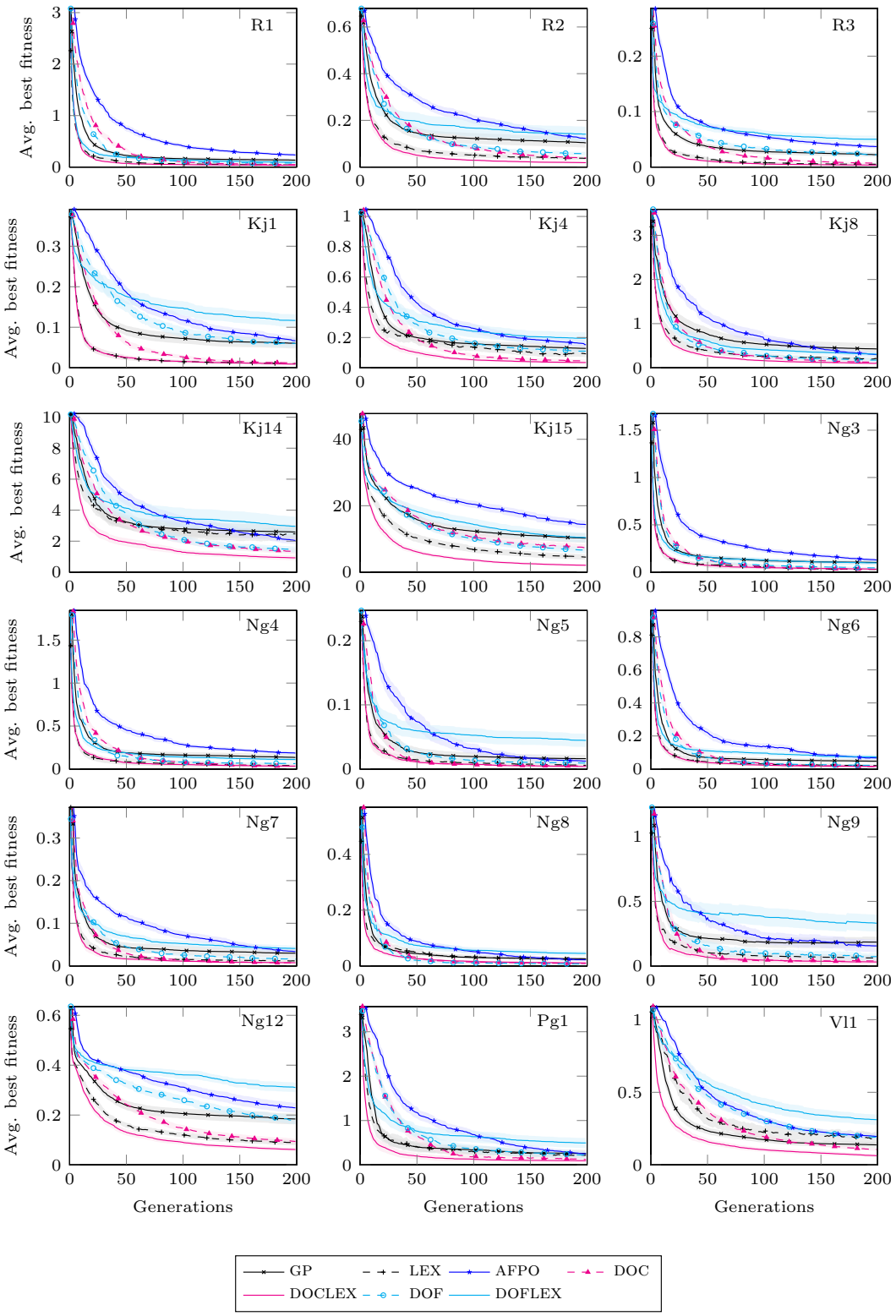


Figure 11.1: Average and .95-confidence interval of the best-of-generation fitness.

Table 11.2: Average and .95-confidence interval of the best-of-run fitness. Last row present the averaged ranks of setups.

Problem	GP	LEX	AFPO	DOC	DOCLEX	DOF	DOFLEX
R1	0.133 ±0.027	0.039 ±0.013	0.235 ±0.025	0.046 ±0.008	**0.034** ±0.011	0.076 ±0.013	0.093 ±0.013
R2	0.104 ±0.020	0.037 ±0.007	0.122 ±0.014	0.040 ±0.006	**0.019** ±0.003	0.057 ±0.008	0.141 ±0.032
R3	0.022 ±0.004	0.005 ±0.001	0.037 ±0.003	0.007 ±0.001	**0.003** ±0.001	0.022 ±0.003	0.050 ±0.006
Kj1	0.060 ±0.013	0.010 ±0.003	0.067 ±0.009	0.012 ±0.003	**0.009** ±0.002	0.060 ±0.010	0.117 ±0.015
Kj4	0.128 ±0.021	0.092 ±0.021	0.152 ±0.022	0.045 ±0.009	**0.034** ±0.006	0.110 ±0.011	0.195 ±0.040
Kj8	0.427 ±0.128	0.208 ±0.048	0.297 ±0.059	0.126 ±0.030	**0.099** ±0.020	0.174 ±0.051	0.306 ±0.056
Kj14	2.586 ±0.463	2.468 ±0.415	2.031 ±0.242	1.354 ±0.182	**0.923** ±0.153	1.457 ±0.196	2.957 ±0.630
Kj15	10.229 ±1.380	4.517 ±0.793	14.236 ±1.189	7.426 ±0.921	**2.069** ±0.324	6.629 ±0.946	10.428 ±1.432
Ng3	0.103 ±0.022	**0.029** ±0.016	0.128 ±0.016	0.030 ±0.005	0.031 ±0.010	0.045 ±0.007	0.097 ±0.024
Ng4	0.135 ±0.035	0.040 ±0.016	0.184 ±0.018	**0.031** ±0.007	0.033 ±0.010	0.064 ±0.012	0.111 ±0.019
Ng5	0.016 ±0.006	0.006 ±0.002	0.012 ±0.003	0.005 ±0.001	**0.004** ±0.001	0.009 ±0.002	0.045 ±0.010
Ng6	0.047 ±0.012	0.021 ±0.007	0.066 ±0.012	**0.015** ±0.005	0.016 ±0.004	0.022 ±0.004	0.073 ±0.013
Ng7	0.030 ±0.010	0.012 ±0.004	0.034 ±0.006	**0.007** ±0.001	0.007 ±0.002	0.016 ±0.003	0.041 ±0.007
Ng8	0.025 ±0.005	0.024 ±0.006	0.022 ±0.005	0.009 ±0.003	0.011 ±0.003	**0.007** ±0.005	0.045 ±0.009
Ng9	0.184 ±0.042	0.061 ±0.022	0.154 ±0.048	0.038 ±0.030	**0.030** ±0.011	0.072 ±0.029	0.331 ±0.065
Ng12	0.184 ±0.021	0.089 ±0.010	0.228 ±0.020	0.094 ±0.011	**0.062** ±0.008	0.180 ±0.017	0.310 ±0.019
Pg1	0.259 ±0.079	0.221 ±0.069	0.247 ±0.046	0.121 ±0.026	**0.094** ±0.042	0.223 ±0.075	0.495 ±0.112
Vl1	0.138 ±0.018	0.189 ±0.042	0.196 ±0.018	0.107 ±0.009	**0.063** ±0.011	0.191 ±0.018	0.311 ±0.036
Rank:	5.444	2.889	5.778	2.222	1.389	3.722	6.556

Table 11.3: Median and .95-confidence interval of test set fitness of the best-of-run programs. Last row presents the averaged ranks of setups.

Problem	GP	LEX	AFPO	DOC	DOCLEX	DOF	DOFLEX
R1	0.192 ±0.042	0.046 ±0.017	0.266 ±0.032	0.055 ±0.009	**0.042** ±0.016	0.071 ±0.012	0.124 ±0.043
R2	0.127 ±0.022	0.059 ±0.017	0.141 ±0.018	0.047 ±0.008	**0.033** ±0.017	0.071 ±0.012	0.184 ±0.045
R3	0.043 ±0.006	0.010 ±0.003	0.052 ±0.005	0.011 ±0.002	**0.009** ±0.002	0.028 ±0.004	0.068 ±0.009
Kj1	0.126 ±0.021	0.071 ±0.027	0.108 ±0.018	**0.031** ±0.009	0.044 ±0.014	0.098 ±0.017	0.180 ±0.027
Kj4	0.555 ±0.097	0.329 ±0.068	0.291 ±0.058	**0.164** ±0.052	0.282 ±0.076	0.274 ±0.052	0.684 ±0.131
Kj8	210.659 ±405.680	29.085 ±37.352	1.029 ±0.307	0.882 ±0.273	3.235 ±3.101	**0.639** ±0.237	1.218 ±0.410
Kj14	3.785 ±0.647	3.040 ±0.541	2.516 ±0.340	**1.548** ±0.179	1.672 ±0.364	2.151 ±0.425	4.264 ±0.855
Kj15	11.380 ±1.391	5.466 ±1.063	14.601 ±1.108	7.367 ±0.831	**3.005** ±0.471	6.885 ±0.910	12.243 ±1.545
Ng3	0.120 ±0.025	**0.033** ±0.017	0.146 ±0.018	0.034 ±0.006	0.034 ±0.010	0.043 ±0.006	0.099 ±0.023
Ng4	0.173 ±0.035	0.049 ±0.019	0.205 ±0.019	**0.035** ±0.010	0.047 ±0.015	0.071 ±0.014	0.133 ±0.020
Ng5	0.024 ±0.010	0.018 ±0.009	0.011 ±0.003	**0.005** ±0.001	0.005 ±0.002	0.009 ±0.002	0.053 ±0.021
Ng6	0.046 ±0.013	0.025 ±0.011	0.037 ±0.011	**0.008** ±0.004	0.016 ±0.005	0.014 ±0.004	0.070 ±0.015
Ng7	0.040 ±0.012	0.018 ±0.007	0.038 ±0.008	**0.012** ±0.005	0.066 ±0.045	0.025 ±0.008	0.051 ±0.011
Ng8	0.129 ±0.018	0.135 ±0.032	0.077 ±0.024	0.056 ±0.016	0.114 ±0.038	**0.024** ±0.015	0.176 ±0.041
Ng9	0.211 ±0.055	0.058 ±0.032	0.079 ±0.038	**0.012** ±0.012	0.131 ±0.088	0.085 ±0.041	0.501 ±0.107
Ng12	0.265 ±0.044	**0.187** ±0.029	0.282 ±0.020	0.188 ±0.031	0.220 ±0.064	0.297 ±0.023	0.448 ±0.048
Pg1	2.085 ±0.277	1.746 ±0.272	1.580 ±0.217	**1.170** ±0.168	1.475 ±0.198	1.717 ±0.222	2.252 ±0.217
Vl1	0.636 ±0.076	0.532 ±0.079	0.608 ±0.086	0.427 ±0.043	**0.427** ±0.075	0.562 ±0.052	0.831 ±0.081
Rank:	5.667	3.333	4.889	1.722	2.667	3.333	6.389

Table 11.4: Post-hoc analysis of Friedman's test conduced on Table 11.2 (training set, left) and 11.3 (test set, right): p-values of incorrectly judging a setup in a row to achieve better fitness than a setup in a column. Significant values ($\alpha = 0.05$) are marked in bold.

	GP	LEX	AFPO	DOC	DOCLEX	DOF	DOFLEX
GP			0.999				0.719
LEX	**0.007**		**0.001**			0.910	**0.000**
AFPO							0.934
DOC	**0.000**	0.969	**0.000**			0.363	**0.000**
DOCLEX	**0.000**	0.363	**0.000**	0.910		**0.020**	**0.000**
DOF	0.201		0.065				**0.002**
DOFLEX							

	GP	LEX	AFPO	DOC	DOCLEX	DOF	DOFLEX
GP							0.953
LEX	**0.020**		0.317				**0.000**
AFPO	0.934						0.363
DOC	**0.000**	0.275	**0.000**		0.847	0.275	**0.000**
DOCLEX	**0.001**	0.969	**0.033**			0.969	**0.000**
DOF	**0.020**	1.000	0.317				**0.000**
DOFLEX							

hypothesis that a setup in a row is better than a setup in a column. The significant p-values are marked in bold. This comparison indicates that the performance improvement of DOC and DOCLEX relative to control methods GP and AFPO is significant.

For a more detailed insight, we also rank all configurations in the bottom row of Table 11.2. The best overall average rank of 1.39 was achieved by DOCLEX, which outperforms the other methods on 13/18 benchmarks. The second is DOC with the average rank of 2.22 and the lowest error on 3/18 benchmarks. Third place is taken by LEX with the average rank of 2.89 and the lowest error on 1/18 benchmarks.

Concerning the generalization capability, Table 11.3 presents the median and 95% confidence interval of test set fitness of the best-of-run program. The results are largely consistent with the

Table 11.5: Average and .95-confidence interval of number of nodes in the best-of-run program. Last row presents the averaged ranks of setups.

Problem	GP		LEX		AFPO		DOC		DOCLEX		DOF		DOFLEX	
R1	92.437	±9.83	100.008	±11.07	**30.170**	±1.38	81.442	±7.02	101.085	±12.20	89.206	±4.57	87.903	±14.13
R2	99.776	±17.78	101.452	±9.66	**25.597**	±1.35	84.166	±6.02	110.236	±14.05	109.446	±6.62	90.809	±8.16
R3	112.906	±14.58	122.057	±10.60	**32.255**	±1.63	103.675	±7.76	123.575	±13.85	134.024	±9.05	122.974	±9.67
Kj1	111.203	±16.66	134.972	±16.18	**32.499**	±1.59	96.185	±9.05	109.394	±12.52	144.953	±10.27	148.799	±13.63
Kj4	133.811	±15.04	130.992	±9.75	**40.949**	±2.22	108.622	±9.68	134.536	±13.96	142.368	±10.57	146.039	±13.20
Kj8	150.517	±16.56	140.043	±11.70	**35.363**	±1.81	95.444	±7.90	147.541	±12.49	117.642	±8.25	131.048	±10.25
Kj14	89.812	±7.69	86.670	±7.10	**23.324**	±1.24	71.025	±3.97	98.103	±9.04	95.905	±9.33	110.179	±8.01
Kj15	110.263	±11.37	116.492	±8.23	**26.498**	±1.36	108.840	±7.48	134.830	±9.14	104.770	±6.94	136.477	±10.61
Ng3	89.343	±11.99	99.310	±12.00	**27.755**	±1.86	65.468	±5.01	100.422	±12.84	100.974	±8.16	94.524	±8.10
Ng4	91.692	±11.22	91.805	±9.54	**27.099**	±1.32	86.014	±9.43	98.851	±9.93	94.838	±6.47	93.499	±7.34
Ng5	60.719	±9.52	66.786	±10.72	**19.564**	±1.63	51.837	±6.40	67.796	±10.52	99.257	±12.23	84.411	±10.61
Ng6	82.674	±11.56	80.698	±7.67	**24.255**	±2.01	69.064	±8.43	83.287	±10.08	113.484	±16.39	96.361	±9.40
Ng7	66.389	±9.14	75.943	±6.52	**19.454**	±1.47	67.261	±5.45	90.062	±10.93	129.159	±15.66	89.042	±9.22
Ng8	63.756	±8.55	80.615	±10.48	**22.530**	±1.26	54.663	±6.22	75.417	±8.86	89.167	±10.35	81.057	±7.61
Ng9	90.664	±14.54	98.202	±13.23	**20.756**	±2.43	64.692	±11.44	87.062	±11.47	104.855	±12.87	98.217	±10.35
Ng12	83.121	±13.88	91.343	±6.95	**22.000**	±1.44	95.799	±6.73	116.942	±11.98	142.845	±14.95	104.046	±15.90
Pg1	64.462	±7.80	87.612	±7.15	**24.062**	±1.12	64.629	±3.85	90.902	±10.21	91.096	±8.08	110.165	±12.05
Vl1	107.003	±10.53	111.494	±10.92	**29.776**	±1.47	100.707	±6.98	126.004	±10.70	120.169	±7.46	121.183	±13.61
Rank:	3.556		4.278		1.000		2.278		5.556		5.833		5.500	

Table 11.6: Post-hoc analysis of Friedman's test on Table 11.5. Significant values ($\alpha = 0.05$) are in bold.

	GP	LEX	AFPO	DOC	DOCLEX	DOF	DOFLEX
GP		0.954			0.080	**0.026**	0.098
LEX					0.565	0.317	0.618
AFPO	**0.007**	**0.000**		0.565	**0.000**	**0.000**	**0.000**
DOC	0.565	**0.040**			**0.000**	**0.000**	**0.000**
DOCLEX						1.000	
DOF							
DOFLEX					1.000	0.999	

results on the training set, and again we observe the positive effects of driving search using derived objectives. Across all problems, the median best fitness on the test sets is obtained by DOC, which achieves the best overall average rank of 1.72. DOCLEX is second with the rank of 2.67, while LEX and DOF both rank third with the same average result of 3.33. The methods do not exhibit heavy overfitting to training data, except for Kj8, Kj15 and Pg1, where higher test errors are apparent. Interestingly, despite this tendency, DOC and DOF manage to maintain the lowest errors on these benchmarks. Friedman's test conducted on test set fitness from Table 11.3 results in p-value of 8.02×10^{-10} and the right inset in Table 11.4 demonstrates its post-hoc analysis. Observations are largely confirmed: DOC and DOF are both better than GP, while DOC and DOCLEX also significantly outperform AFPO.

Table 11.5 shows the average and 95% confidence interval of the number of nodes in the best-of-run programs. We are not surprised to see AFPO produce the smallest programs on average, as one of its motivations was to address the issue of bloat in GP. DOC comes second, with the rank of 2.27, and produces much smaller programs than the control methods. LEX, which ranked next after DOC in terms of fitness, turns out to produce much larger programs, even though its average run lengths do not diverge much from those of DOC (the run lengths are not reported for brevity). This observation is confirmed by the Friedman's test shown in Table 11.6 – DOC produces significantly smaller programs than LEX, DOCLEX, DOF and DOFLEX. LEX also seems to have slightly detrimental effect on program size when used in a hybrid approach with DOC.

11.3 Discussion

The overall positive results corroborate our findings from the previous chapters and extend them to continuous domains. The alternative, transient objectives derived automatically from interaction matrices by DOC and DOF turn out to surpass a range of other methods on the key performance

indicators. Crucially, the proposed methods manage to outperform also LEX, which showed superior performance in multiple previous studies [127, 197, 218].

This outcome suggests also that the decisions we made when designing the preprocessing method described in Section 11.1 were largely appropriate. Indeed, preliminary experiments, not reported here for brevity, suggested that all key components of that method are essential:

- The 'capping' of the maximum error with φ in Step 2 to emphasize the differences in low ranges of error,
- The standardization in Step 3 to provide for the same magnitude of outcomes on tests, while maintaining their individual capability of discrimination between programs, and
- The sigmoid squeezing in Step 4 to map the outcomes in entire matrix to the same interval so that they have same magnitude and are thus comparable when used by DOC and DOF.

The method is also effectively parameter-free, as it adjusts φ automatically, relying on the statistics of error distribution in G.

It is encouraging to see that the low errors on the training set in most cases translate to test sets. Given that smaller programs tend to generalize better, this result can be in part attributed, particularly for DOC and DOCLEX, to moderate sizes of programs evolved by these methods. This is interesting, given that, except for AFPO, none of the setups considered here explicitly rewards smaller programs. It would be interesting to find out whether there are any other (than size) characteristics of the programs evolved with derived objectives that make them perform so well, and we consider this one of interesting follow-up directions.

Last but not least, let us note that the overall underperformance of DOFLEX should be mainly attributed to the fact that we aimed here at possibly non-arbitrary parameter settings. For DOF, no obvious means for automatic setting of factorization rank have been proposed to date, so we set it to the highest possible value ($r = n$), as there is substantial conceptual and empirical evidence that LEX yields the best results when the number of tests is at least in dozens [127, 128]. However, it seems that this setting is suboptimal for DOF, leading to discovery of non-discriminative and redundant objectives. One possible reason for such a behavior is the observation that NMF tends to set many factors to zero for high values of r, which is not necessary beneficial from the perspective of discovery of search objectives. This observation points to another natural follow-up research, which could investigate the role of sparsity.

11.4 Chapter summary

In this chapter, we empirically generalized the findings from Chapter 9 and 10 concerning methods that derive search objectives from interaction matrices. We may thus claim now that derived objectives, though heuristically derived and transient, are effective means of search not only for domains with discrete and/or constrained interaction outcomes, but also for those where interaction outcomes are continuous and unconstrained. This elevates the derived objectives to a fully-fledged concept for metaheuristics applied to test-based problems, including, but not limited to, GP. Given that most of the proposed methods are practically parameter-free, and so is the preprocessing method proposed in this chapter, one may seriously consider including these solutions as out-of-the-box functionalities in metaheuristic toolkits.

Chapter 12

Surrogate Fitness via Factorization of Interaction Matrix

Evaluation of candidate solutions in evolutionary computation is usually the most expensive component of search process in terms of computation time required. This is particularly true in applications where fitness evaluation involves some form of simulation, e.g., of an engineering artefact (like a controller) in some virtual environment. Evaluation may become even more expensive in test-based problems where assessing a candidate solution requires simulating its behavior in *multiple* contexts. This is often necessary in order to account for the inherent noisiness of the environment, or to address the possibility of multiple initial states, or in adversarial environments where the context is actively opposing the agent (solution). Two- and multi-player games are good examples of the latter case. Arguably, testing a program on a given set of tests in GP can also be seen as an expensive simulation, particularly when programs become large.

Lowering the computational cost incurred by evaluation by simply reducing the number of tests is often not a viable option. Few tests implies inaccurate fitness, and consequently a poorly informed search process. Moreover, discarding tests may cause a task to be formally underspecified (underconstrained). For instance, a set of tests for a multiplexer problem (see Example 6.1) that misses even a single test does not technically specify that problem anymore. Also, when an interaction function is binary, a low number of tests leads to coarse-grained fitness that often fails to differentiate solutions (cf. Section 6.1).

The practical answer to this challenge in evolutionary computation are surrogate models (cf. Section 8.6), in which the original fitness function $f : X \to \mathbb{R}$ is abandoned in favor of a computationally less expensive, albeit approximate, *surrogate fitness function* $\hat{f} : X \to \mathbb{R}$. In some cases, $\hat{f}$ can be designed and implemented manually by an expert knowledgeable in the given application area. However, it is often more convenient and became recently more common to *learn* such models from examples using machine learning algorithms [58].

In this chapter, we demonstrate that the conceptual framework for discovery of search objectives comes in handy also in such a setting. The formalism of NMF that is the core component of DOF algorithm (cf. Chapter 10) allows not only deriving search objectives, but also *predicting* outcomes of interactions between candidate solutions in S and tests in T. In the following, we propose a method inspired by this observation that heuristically estimates the fitness from G using NMF. Crucially, that estimation requires only a fraction of elements of G to be known, which implies that only some interactions between candidate solutions and tests have to be conducted. As we demonstrate in the following, this allows substantial reduction of required computational effort.

The method presented in this chapter has been originally published [212] and later extended in [217, 210].

12.1 Factorization of G with missing interaction outcomes

Recall from Section 10.1 that NMF produces two non-negative matrices W and H that multiplied together form a lower rank approximation of G. To this end, NMF solves optimization problem given by (10.1.2). For reader's convenience, we repeat it here

$$\min_{W,H} L(W,H) \equiv \frac{1}{2}||G - WH||_F^2 \;\; s.t. \;\; W,H \geq 0. \tag{12.1.1}$$

Crucially for our further considerations, G can be factorized in the this way *even if some of its elements are unknown*, i.e., when G is *sparse*. This makes matrix factorization a powerful tool in *collaborative filtering* (a technique used by recommender systems [298]), where it can be used to fill in the gaps in a large matrix (of, e.g., users' recommendations [169]), even if only small part of that matrix is known for certain.

As it follows from (10.1.1), to predict an interaction outcome of a candidate solution s with a test t from the matrices W and H found by solving (12.1.1), we calculate the dot product of two vectors corresponding to s and t:

$$\hat{g}_{st} = \mathbf{w}_s^T \mathbf{h}_t = \sum_{j=1}^{r} w_{sj} h_{jt}. \tag{12.1.2}$$

However, the update rules given by (10.1.3) and (10.1.4) implicitly assume that the input matrix is complete. For sparse matrices, arguably the most popular approach to minimize expression (12.1.1) is stochastic gradient descent (SGD) optimization. The algorithm loops through all available interaction outcomes in G in random order and for each given training case g_{st} it first predicts $\hat{g}_{st}$ and then computes the associated prediction error between the known and predicted outcome of interaction for given s and t

$$e_{st} = g_{st} - \hat{g}_{st} = g_{st} - \mathbf{w}_s^T \mathbf{h}_t.$$

Then it modifies the parameters proportionally to the learning rate γ in the opposite direction of the gradient, i.e.:

$$w_{sj} = w_{sj} - \gamma \frac{\partial}{\partial w_{sj}} e_{st}^2 = w_{sj} + \gamma e_{st} h_{jt},$$

$$h_{jt} = h_{jt} - \gamma \frac{\partial}{\partial h_{jt}} e_{st}^2 = h_{jt} + \gamma e_{st} w_{sj},$$

where $j = 1, \ldots, r$. To prevent overfitting to the interaction outcomes used for training, its common to add the regularization term $\lambda(||W||_F^2 + ||H||_F^2)$ to the loss function, which yielding the following update rules:

$$w_{sj} = w_{sj} + \gamma(e_{st} h_{jt} - \lambda w_{sj}), \tag{12.1.3}$$

$$h_{jt} = h_{jt} + \gamma(e_{st} w_{sj} - \lambda h_{jt}), \tag{12.1.4}$$

The update rules are applied for a fixed number of iterations, or until the error given by (12.1.1) is sufficiently small. Stochastic gradient descent owes its popularity to the ease of implementation combined with fast running times. For a comparison of various NMF techniques for collaborative filtering see e.g. [109].

Crucially for our needs, stochastic approach helps exploiting behavioral similarity between candidate solutions and tests in an interaction matrix because parameter updates based on interaction outcomes from a certain row or column will also decrease the loss in similar rows and columns. Furthermore, gradient-based NMF algorithms usually converge in at most a few dozens of steps, even when G is relatively large [169, 28].

Algorithm 8 Surrogate Fitness via Factorization of Interaction Matrix (SFIMX).

Require: factorization rank r.

1: **function** SFIMX(S, T, α)
2: **for** $s \in S$ **do**
3: $T' \leftarrow \text{SAMPLE}(T, \alpha)$
4: **for** $t \in T'$ **do**
5: $g(s,t) \leftarrow \text{INTERACT}(s,t)$ ▷ apply candidate s to test t
6: $W, H \leftarrow \text{NMF}(G, r)$
7: $\hat{G} \leftarrow WH$ ▷ predict missing g_{ij}s
8: $\hat{G} \leftarrow \text{REPLACEKNOWN}(G, \hat{G})$
9: **for** $s_i \in S$ **do**
10: $f(s_i) \leftarrow \sum_{j=1}^{n} \hat{g}_{ij}$
11: **return** F

12.2 SFIMX

As argued in the introduction to this chapter, computing a complete interaction matrix G can be computationally costly, particularly if executing individual interactions g_{ij} is expensive, the number of tests in T is large, or both. To alleviate this problem, we propose a method dubbed Surrogate Fitness via Factorization of Interaction Matrix (SFIMX), which calculates only a fraction of interaction outcomes (so that G is sparse), and uses NMF to estimate the remaining interaction outcomes from those known ones. The method expects two parameters: the factorization rank r and the desired density $\alpha \in (0, 1]$ of partial interaction matrix (meant as a complement of sparsity). SFIMX employs the NMF formalisms described in Sections 10.1 and 12.1 to replace the conventional fitness evaluation stage of EA with the following steps:

1. Calculate *in part* the sparse $m \times n$ interaction matrix G between the candidate solutions from the current population S and the tests from T in the following way:

   a) For each candidate solution s, draw a nonempty random subset of tests $T' \subset T$ of size αn to interact with, where $\alpha \in (0, 1]$ is the parameter that controls the fraction of interactions to be calculated.

   b) Perform interactions between s and tests in T', placing their outcomes in the appropriate cells of the corresponding row of G.

2. Factorize G in non-negative components W and H, using only the known elements of G, and ignoring any missing (unknown) interaction outcomes.

3. Use the obtained matrices to estimate all interaction outcomes by calculating $\hat{G} = WH$.

4. Replace predictions in $\hat{G}$ with interaction outcomes that are given in G.

5. Compute from $\hat{G}$ the fitness of a candidate solution $s \in S$ as:

$$f_{SFIMX}(s) = \sum_{j=1}^{n} \hat{g}_{sj}, \tag{12.2.1}$$

   i.e. in the same way as in the conventional scalar fitness.

These steps are summarized in Algorithm 8. To comply with NMF's requirement of G's elements being strictly positive, in line 5 of Algorithm 8, we set $g(s,t) = 1$ if s fails t and $g(s,t) = 2$ otherwise. Note that $\alpha \geq \frac{1}{|T|}$ must hold for T' to be nonempty.

Example 12.1. Consider the population of candidate solutions $S = \{s_1, s_2, s_3, s_4\}$ and the set of tests $T = \{t_1, t_2, t_3, t_4, t_5\}$. Assume that SFIMX is run with $\alpha = \frac{3}{5}$ and step 1 of the above algorithm yields the following sparse matrix of interactions G between S and T:

$$G = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{pmatrix} 2 & & 1 & 2 & \\ & 2 & 1 & & 1 \\ 1 & & & 2 & 2 \\ 2 & 1 & & & 1 \end{pmatrix} \end{array}$$

Let $r = 3$. The application of NMF implemented via SGD to G (line 6 of Algorithm 8) returns the following decomposition into W and H:

$$W = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{ccc} f_1 & f_2 & f_3 \end{array} \\ \begin{pmatrix} 0.46 & 1.96 & 0.6 \\ 1.27 & 0.1 & 0.95 \\ 1.37 & 0.02 & 2.83 \\ 0.4 & 1.86 & 1.60 \end{pmatrix} \end{array}, \quad H = \begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{pmatrix} 0.48 & 1.50 & 0.01 & 0.41 & 0.41 \\ 0.87 & 0.14 & 0.19 & 0.77 & 0.01 \\ 0.11 & 0.09 & 1.02 & 0.50 & 0.51 \end{pmatrix} \end{array}.$$

When multiplied (line 7 of Algorithm 8), W and H lead to the following reconstructed interaction matrix:

$$\hat{G} = WH = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{pmatrix} 2 & 1.02 & 1 & 2 & 0.52 \\ 0.8 & 2 & 1 & 1.07 & 1 \\ 1 & 2.31 & 2.1 & 2 & 2 \\ 2 & 1 & 2.01 & 2.4 & 1 \end{pmatrix} \end{array}$$

In the next step, ReplaceKnown copies the known interaction outcomes from G to $\hat{G}$ (this step is ineffective in this example, as the outcomes for known interactions have been perfectly estimated, but that is not always the case). Finally, in line 10, we calculate the fitness of particular candidate solutions by summing the corresponding rows of the reconstructed interaction matrix, which results in $f(s_1) = 6.54$, $f(s_2) = 5.87$, $f(s_3) = 9.41$, and $f(s_4) = 8.41$. Overall, SFIMX enabled calculating these values using $\alpha nm = 12$ known interaction outcomes, compared to $nm = 20$ interactions required by the conventional scalar fitness function. ■

In the above example, the reconstructed matrix $\hat{G}$ perfectly reproduces the known interaction outcomes, so the square approximation error minimized by NMF (Eq. 10.1.2) attains zero. In general, the approximation error tends to be greater for smaller r and greater α. It should be noted however that, regardless how well NMF reconstructs the known interaction outcomes, the unknown ones are only extrapolated, so the value of SFIMX's fitness will in general diverge from the true fitness (Eq. 5.5.1).

12.3 Properties of SFIMX

SFIMX is designed for test-based problems (cf. Section 5). Given that in many instances of problems belonging to this class individual tests come from the same domain and together describe the desired behavior of the same input-output functionality, one can expect the outcomes of candidate solution's interactions with them to be mutually dependent. This condition is necessary for SFIMX to accurately predict the outcomes of some interactions from the outcomes of other interactions. Nevertheless, we expect it to be satisfied in many practical situations, as it is actually

much harder (if not impossible) to come across problems where interaction outcomes are entirely independent. Notice also that if this was the case, then solving every test would require from a candidate solution completely different set skills. This is however in conflict with the nature of learning tasks which typically involve discovery of some sort of general knowledge that has to be extrapolated even beyond the training cases.

Predictions made by SFIMX are based on how behaviorally similar candidate solutions interact with the tests in T. As a result, evaluation in SFIMX is *contextual*: a prediction $\hat{g}_{st}$ made for a missing outcome depends not only on s and t, but also on other candidate solutions in S and other tests in T. Consequently, all available outcomes of interactions between S and T together determine the NMF model and therefore influence how predictions for missing interaction outcomes are made.

SFIMX's estimates are linear combinations of the elements in W and H. Furthermore, the non-negativity constraint allows only additive linear combinations of the factors, thus enabling it to learn 'parts' that have distinct features and physical representations [206]. Notice also that interaction outcomes reconstructed by NMF are real-valued rather than discrete. As a result, even when the known interaction outcomes are binary (1s and 2s), the fitness calculated by SFIMX (Eq. 12.2.1) is no longer restricted to $n+1$ discrete values, but can assume an arbitrary value. That leads to a more fine-grained fitness that is likely to be less susceptible to loss of gradient (cf. Section 6.3).

SFIMX cannot accumulate any knowledge about the characteristics of individual tests along an evolutionary search process – at its heart it is a *memoryless* estimation technique that starts anew in each generation. While this may appear like a disadvantage, we argue that it actually benefits SFIMX, due to the capability to immediately react to the ongoing changes in the population. Such an approach also precludes any past bias that could otherwise negatively impact the current estimates.

One might argue that SFIMX does not diverge so much from the traditional approaches to surrogate fitness, and that the only essential difference is that the former estimates the outcomes of *interactions*, while the latter estimate the outcomes of *fitness evaluation*. In response, let us point to an important advantage of our method, which it owes to the tight embedding in test-based problems: SFIMX predicts interaction outcomes *from other interaction outcomes* (and by this token also from *behavior* of candidate solutions on tests), and in this way abstracts from the characteristics of a given application. In contrast, traditional surrogates predict fitness from the (usually genotypic) properties of candidate solutions, and are thus more application-specific and more difficult to design.

12.4 Experiment

The purpose of the experiment is twofold. Firstly, we examine the performance of SFIMX in the domain of tree-based GP, following the experimental protocol and benchmark problems used in Section 9.5. Secondly, we are interested in verifying whether SFIMX is a viable method for reducing the computational cost incurred by evaluation. For that aim, we control the fraction of interactions to be calculated by the parameter $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$ in SFIMX algorithm (cf. Section 12.2).

The factorization is realized by the SGD algorithm ((12.1.3) and (12.1.4)). As in Section 10.5, we perform up to 50 iterations of SGD, each involving both steps, i.e., (12.1.3) and (12.1.4). If the approximation error (Eq. 10.1.2) drops below 10^{-5}, we stop the optimization earlier.

12.4.1 Compared algorithms

We consider three settings of factorization rank r that controls the degree to which the interaction outcomes are being compressed by factorization. The configuration dubbed **SFIMX-F** uses $r = \min(m, n)$, which is equivalent here to $r = n$, because for the considered benchmarks $m > n$. This value should be considered large, as NMF can then perfectly reproduce the known interaction outcomes, because the rank of G can be at most $\min(m, n)$.

The **SFIMX-H** configuration uses $r = n/2$, which forces the interaction outcomes to be compressed in half the number of weights in matrix W and features in matrix H. However, this number can be still considered quite high, given that we expect the interaction outcomes to be mutually correlated between candidate solutions and tests.

Finally, the configuration **SFIMX-L** uses the smallest rank $r = \lceil \log_2 n \rceil$. In this case, r is in the order of the number of input variables; for instance, for the Mux6 problem $r = \log_2 2^6 = 6$.

The non-SFIMX baseline methods include conventional Koza-style GP (cf. Section 9.5.1) and Random Subset Selection (RSS). The latter calculates fitness using a subset of αn tests drawn independently (and anew in every generation) for each row of G. This proceeding is intended to mimic the evaluation scheme known in coevolutionary algorithms [53].

To verify whether an ideal program has been found, in the last generation of an evolutionary run, we evaluate the best candidate solution(s) on all tests. This amounts to computing additional $(1 - \alpha)n$ interactions and applies to all SFIMX and RSS configurations.

12.4.2 Success rates

We focus on two aspects in the analysis that follows: SFIMX's end-of-run success rate and the accuracy of predicting interaction outcomes. The first one reflects the end-to-end performance of the proposed method, most relevant from the practical perspective of solving discrete-fitness test-based problems (here: program synthesis). The second aspect characterizes SFIMX on a more internal and elementary level. For the purpose of this experiment, we limit our attention to SFIMX-F for $\alpha \in [0.1, 0.9]$ (denoted as SFIMX-F$_\alpha$). The results presented below are averages over 50 independent runs of evolution, repeated for each combination of method and problem.

Table 12.1 shows the success rates obtained by particular variants of methods on each benchmark, as a function of the fraction α of the performed interactions. The last column contains the global rank of a given configuration, obtained by averaging the ranks on individual benchmarks. The last row shows success rates obtained by ordinary GP.

The best overall average rank of 6.03 is achieved by SFIMX-F$_{0.4}$. Eight out of nine SFIMX-F configurations rank above any of the control configurations; only SFIMX-F$_{0.1}$ ranks below GP and some RSS setups. The last observation is not surprising, given that SFIMX-F$_{0.1}$ is forced to make predictions from the mere 10 percent of actual outcomes of program-test interactions. GP attained the average rank of 8.78, surpassing all RSS configurations.

It is difficult to establish a clear pattern as to which values of α are the most beneficial in terms of success rates. As a rule of thumb, we may say that setting α in $[0.4, 0.6]$ is favorable. However, using other values is not very detrimental, except for $\alpha = 0.1$ that should be avoided. For many problems SFIMX-F maintains decent success rates even for very low setting of this parameter; for instance it is better than or comparable to GP for $\alpha = 0.2$ on Cmp8, Mux6, Dsc1, Dsc2, Dsc3, Dsc5, Mal2, Mal3, Mal4, and Mal5. On the other hand, we also anticipate that too high values of α may be also suboptimal. Our hypothesis is that high α reduces the amount of noise in fitness estimates, and some noise in fitness has been proven helpful in past works [53].

Table 12.1: Success rate of SFIMX-F (×100) of best-of-run individuals, averaged over 50 evolutionary runs. Bold marks the best result for each benchmark.

α	Cmp6	Cmp8	Maj6	Maj8	Mux6	Par5	Dsc1	Dsc2	Dsc3	Dsc4	Dsc5	Mal1	Mal2	Mal3	Mal4	Mal5	Rank
								SFIMX-F									
0.9	44	2	40	0	100	0	0	4	46	0	16	80	64	84	40	98	6.12
0.8	66	0	68	0	98	**4**	0	0	42	0	4	86	62	80	30	88	7.53
0.7	**68**	0	64	0	98	0	4	0	48	0	8	78	57	82	34	94	6.69
0.6	50	2	54	0	**100**	0	2	0	44	0	4	76	57	**90**	50	92	6.59
0.5	44	0	50	0	**100**	0	2	8	44	0	2	72	**70**	82	**52**	**100**	6.53
0.4	46	**10**	**70**	0	**100**	2	4	6	52	0	10	76	36	74	26	82	6.03
0.3	36	0	44	0	**100**	0	**10**	12	42	0	**16**	60	48	88	42	90	7.25
0.2	36	0	32	0	**100**	0	4	**16**	**64**	0	2	70	66	80	36	94	7.31
0.1	12	0	2	0	90	0	0	0	26	0	0	60	64	62	22	78	13.00
								RSS									
0.9	23	3	56	0	96	0	0	3	26	0	0	73	13	76	3	86	10.94
0.8	30	0	53	0	96	0	0	0	26	0	0	76	50	80	6	90	10.78
0.7	26	0	43	0	86	0	0	0	26	0	3	83	30	46	10	86	12.06
0.6	40	0	46	0	96	0	0	0	23	0	0	56	33	73	16	93	11.44
0.5	30	0	36	0	96	0	0	3	6	0	0	46	26	63	10	96	12.22
0.4	20	0	40	0	86	0	0	0	16	0	6	43	33	66	3	93	12.66
0.3	20	0	16	0	90	0	0	0	6	0	0	53	23	63	0	80	14.44
0.2	3	0	3	0	96	0	0	0	3	0	0	50	30	63	3	66	14.47
0.1	0	0	3	0	66	0	0	0	0	0	0	26	36	30	3	76	15.16
								GP									
GP	54	2	54	0	98	2	0	0	44	0	12	**88**	0	68	0	88	8.78

To provide an aggregated perspective on SFIMX-F's performance, we employ the Friedman's test for multiple achievements of multiple subjects [159] on the best-of-run fitness. The p-value is 1.86×10^{-5}, which strongly indicates that at least one method performs significantly different from the remaining ones. The post-hoc analysis using the symmetry test [135] indicates that the leading configuration SFIMX-$F_{0.4}$ is significantly better than RSS that uses $\alpha \in [0.1, 0.5]$. As suggested by the very close ranks, there is no statistically significant difference between GP and any configuration of SFIMX-F.

On one hand, the above empirical evaluation gives us strong evidence that it is often not sufficient to guide search process using only a sample of tests. On the other, it suggests that SFIMX-F manages to learn patterns in G that allow it to maintain performance on a par with, if not better than, conventional approach that calculates the exact fitness based on all tests. In order to scrutinize SFIMX-F in that aspect, in Table 12.2 we report the average prediction accuracy obtained by SFIMX-F on each benchmark and for each value of α. Technically, the accuracy is the 1-complement of mean absolute error of $\hat{G} = WH$ with respect to G, calculated only for the matrix entries that are being predicted. To that aim, when conducting these evolutionary runs, we calculate also the missing exact interaction outcomes (but use them only for the purpose of this measurement). To complete the picture, the last three rows of the table present the accuracies obtained by SFIMX-F, SFIMX-H and SFIMX-L, averaged over all benchmarks.

The overall observation that emerges from the above experiment is that SFIMX manages to maintain remarkably high level of accuracy, particularly for $\alpha \geq 0.7$ The plausible explanation for the convergence of SFIMX's performance for the higher values of α is that with the majority of interaction outcomes known for certain, predicting the remaining ones (i) becomes relatively easy and can be done well enough using NMF, and/or (ii) the error committed on estimating the outcomes for the remaining interactions is not so critical as to deceive the search process. Individual values in Table 12.2 clearly show that the accuracy of methods' predictions systematically improves with the amount of provided information (i.e., known interaction outcomes). And conversely: in

Table 12.2: Prediction accuracy obtained by SFIMX-F (SX-F$_\alpha$) on each benchmark and for each value of α, averaged over 50 evolutionary runs. The last three rows present the accuracies obtained by SFIMX-F, SFIMX-H and SFIMX-L, averaged over all benchmarks. Bold marks the best result for each benchmark.

Problem	SX-F$_{0.9}$	SX-F$_{0.8}$	SX-F$_{0.7}$	SX-F$_{0.6}$	SX-F$_{0.5}$	SX-F$_{0.4}$	SX-F$_{0.3}$	SX-F$_{0.2}$	SX-F$_{0.1}$
Cmp6	**98.4** ±0.0	98.2 ±0.0	97.7 ±0.0	97.4 ±0.0	96.8 ±0.0	95.8 ±0.1	94.0 ±0.1	91.5 ±0.1	86.8 ±0.2
Cmp8	**99.0** ±0.0	98.9 ±0.0	98.8 ±0.0	98.6 ±0.0	98.4 ±0.0	98.0 ±0.0	97.3 ±0.0	96.4 ±0.1	93.7 ±0.1
Maj6	**97.7** ±0.0	97.4 ±0.0	96.8 ±0.1	96.3 ±0.1	95.5 ±0.1	94.2 ±0.1	92.1 ±0.1	89.0 ±0.2	84.1 ±0.2
Maj8	**98.4** ±0.0	98.3 ±0.0	98.1 ±0.0	97.8 ±0.0	97.2 ±0.0	96.6 ±0.1	95.4 ±0.1	93.3 ±0.1	88.1 ±0.1
Mux6	**97.9** ±0.0	97.1 ±0.0	96.8 ±0.0	95.7 ±0.0	93.8 ±0.1	92.7 ±0.1	86.4 ±0.1	83.6 ±0.2	75.3 ±0.2
Par5	**96.5** ±0.1	95.0 ±0.1	93.0 ±0.2	90.2 ±0.2	85.2 ±0.3	78.3 ±0.3	69.4 ±0.2	55.3 ±0.1	49.3 ±0.0
Dsc1	94.2 ±0.2	**96.3** ±0.2	94.1 ±0.2	94.0 ±0.2	91.8 ±0.2	86.7 ±0.2	80.3 ±0.2	75.1 ±0.1	69.4 ±0.1
Dsc2	**93.6** ±0.1	93.3 ±0.1	92.9 ±0.2	91.0 ±0.2	89.3 ±0.2	86.0 ±0.2	81.1 ±0.2	75.2 ±0.1	67.3 ±0.1
Dsc3	94.6 ±0.1	**94.7** ±0.1	93.5 ±0.1	91.8 ±0.1	89.1 ±0.2	87.0 ±0.2	82.6 ±0.2	77.0 ±0.2	70.2 ±0.1
Dsc4	**98.1** ±0.1	97.6 ±0.1	97.1 ±0.2	96.5 ±0.2	96.1 ±0.2	92.6 ±0.2	85.3 ±0.2	77.0 ±0.1	68.3 ±0.1
Dsc5	**94.1** ±0.2	92.7 ±0.2	92.3 ±0.2	89.7 ±0.2	86.3 ±0.2	80.8 ±0.2	77.8 ±0.2	71.9 ±0.1	66.4 ±0.1
Mal1	**76.2** ±0.3	70.5 ±0.3	69.4 ±0.3	69.4 ±0.3	66.2 ±0.3	61.9 ±0.2	60.0 ±0.2	57.2 ±0.1	55.2 ±0.1
Mal2	**86.0** ±0.2	81.0 ±0.3	76.3 ±0.3	71.3 ±0.3	69.7 ±0.3	63.4 ±0.2	61.1 ±0.2	57.5 ±0.1	54.6 ±0.1
Mal3	**88.7** ±0.2	86.8 ±0.2	80.4 ±0.2	79.2 ±0.3	75.3 ±0.2	70.1 ±0.2	66.5 ±0.2	62.8 ±0.2	59.3 ±0.1
Mal4	**87.8** ±0.2	81.1 ±0.3	81.3 ±0.3	73.4 ±0.3	73.8 ±0.3	68.2 ±0.3	64.6 ±0.2	58.8 ±0.1	56.6 ±0.1
Mal5	**80.2** ±0.3	69.3 ±0.3	65.7 ±0.3	63.2 ±0.3	61.9 ±0.2	57.4 ±0.1	57.0 ±0.1	57.0 ±0.1	53.4 ±0.0
SFIMX-F:	92.6	90.5	89.0	87.2	85.4	81.9	78.2	73.7	68.6
SFIMX-H:	92.3	91.7	89.6	88.0	85.8	80.7	75.3	71.1	66.7
SFIMX-L:	92.3	92.2	91.1	90.2	88.9	85.7	80.0	72.6	65.4

case of SFIMX-F$_{0.1}$, the accuracy drops below 60 percent on 6 benchmarks, and below 70 percent on 10 benchmarks, which may be the main reason for its poor performance.

Interestingly, for Mal benchmarks, the accuracy of SFIMX, is noticeably lower regardless of α. This may suggest that SFIMX lacks the necessary capabilities to capture complex dependencies between interaction outcomes that emerge when solving these problems. This could be the effect of programs in the population becoming more sophisticated and thus starting to exhibit more complex behaviors that is difficult to model using direct linear combination of the elements in W and H. Recall also that Mal benchmarks feature the lowest number of tests, 15, which may make discovering dependencies more difficult.

When juxtaposed, the average accuracies obtained for different values of NMF's rank r reveal that SFIMX-L tends to provide the overall highest accuracy; it outperforms SFIMX-F and SFIMX-H on 6 out of 9 settings of α, only slightly losing in extreme cases, i.e. when α is 0.1, 0.2 or 0.9. Nevertheless, the differences between the methods never exceed 5.3 percent and typically oscillate around 2-3 percent. The likely reason for inferior predictive performance of SFIMX-F and SFIMX-H is overfitting. Both configurations have significantly more parameters to learn, allowing better fit to training data (i.e., the known interaction outcomes), but potentially yielding worse generalization performance (i.e., accuracy of predictions of the unknown

We demonstrate also that SFIMX goes hand in hand with reduced runtime of search. It should be rather obvious that lower values of α lead to greater acceleration of search, positively influencing the overall method's runtime. However, the actual gains depend on the cost of a single interaction and the efficiency of NMF. We may thus anticipate the greatest payoff in applications where interaction function involves some form of time-costly simulation, e.g., of an engineering artifact (like a controller) in some virtual environment. The empirical results shown in Table 12.3 largely confirm our expectations. For brevity, we present only the 'odd' values of α. SFIMX-F$_{0.1}$ takes on average the least time to complete its runs, followed by configurations employing higher values of α. The difference between SFIMX-F$_{0.9}$ and GP reaches roughly 18 percent on average and stems

Table 12.3: Runtimes (in seconds) of SFIMX-F (SX-F$_\alpha$) on each benchmark and for each value of α, averaged over 50 evolutionary runs. Algorithm runtimes (in seconds) for the coevolutionary benchmarks, averaged over 50 evolutionary runs.

Problem	SX-F$_{0.1}$		SX-F$_{0.3}$		SX-F$_{0.5}$		SX-F$_{0.7}$		SX-F$_{0.9}$		GP	
Cmp6	606.2	±24.9	681.7	±51.6	755.7	±90.3	786.1	±69.4	795.8	±91.0	1663.1	±119.7
Cmp8	1131.8	±73.2	1219.2	±51.7	1262.1	±112.0	1309.7	±62.1	1554.1	±75.3	1636.2	±207.1
Maj6	629.9	±46.1	725.3	±48.2	836.8	±108.6	909.6	±59.2	955.0	±82.8	1035.4	±205.4
Maj8	1249.3	±115.3	1501.6	±128.3	1557.6	±91.1	1657.0	±86.7	1709.8	±75.3	2018.1	±269.9
Mux6	652.1	±70.6	767.1	±88.5	807.9	±49.1	910.3	±68.3	1007.8	±58.1	1100.1	±208.3
Par5	213.7	±8.3	454.7	±95.2	750.1	±58.8	881.7	±74.7	956.8	±53.4	1219.1	±132.7
Dsc1	366.9	±19.3	468.5	±56.2	536.4	±148.2	662.4	±89.1	772.9	±155.2	908.9	±47.6
Dsc2	374.3	±25.2	462.8	±102.9	523.4	±41.1	696.6	±72.5	748.6	±94.8	855.7	±86.6
Dsc3	391.6	±56.2	487.6	±26.0	505.6	±22.1	728.5	±64.6	722.4	±46.0	950.6	±47.3
Dsc4	363.4	±46.3	462.1	±20.8	560.4	±89.7	660.2	±98.5	760.9	±84.2	895.5	±78.1
Dsc5	364.3	±85.3	475.0	±111.0	553.1	±126.9	661.3	±129.2	771.1	±156.4	878.8	±152.7
Mal1	264.0	±71.9	391.2	±61.8	591.7	±71.1	613.1	±96.7	705.2	±70.2	817.0	±69.6
Mal2	263.7	±43.8	460.7	±66.1	583.6	±43.5	677.2	±52.6	736.9	±54.1	837.6	±41.5
Mal3	218.3	±48.9	408.4	±53.2	550.6	±38.3	599.2	±38.1	721.2	±37.7	827.0	±26.4
Mal4	261.2	±29.9	449.0	±84.5	552.9	±73.8	607.1	±54.5	650.4	±53.3	913.0	±50.7
Mal5	262.2	±41.6	372.0	±34.9	491.6	±73.2	665.7	±20.1	702.3	±29.3	877.6	±76.3

most likely from bloated programs in GP that consume more time to evaluate. Overall, for αs in the above-recommended interval $[0.4, 0.6]$, SFIMX yields speedups of 34 percent on average.

These results corroborate our hypothesis that that the problem of predicting interaction outcomes in test-based problems can be effectively and efficiently addressed with NMF. By maintaining success rate on par with GP and simultaneously searching the space of candidate solutions faster, SFIMX appears to be a useful surrogate model to speed up the evaluation process.

12.4.3 Results for increased population size

The last section demonstrated that SFIMX manages to successfully reduce the number of program-test interactions necessary for evaluation, thus sparing in each generation $(1-\alpha)mn$ interactions and reducing run time. In the following, we investigate what can be gained by investing these savings in an increased population size. To this end, we increase the baseline population size $m = 1000$ by the factor of $1/\alpha$ (this applies also to RSS). This implies that, for instance, for the lowest α considered here, i.e., 0.1, we increase the population size 10 times. As a result, the overall computational budget (as measured with the number of interactions, i.e., excluding the cost of NMF) is the same for SFIMX and the baseline configurations in both settings, and amounts to $1{,}000n$ interactions per generation and thus $200{,}000n$ interactions per run.

Table 12.4 shows the success rates obtained by particular variants of methods on each benchmark, as a function of the fraction α of the performed interactions, and for three different values of r. The last column (Rank$_\mathrm{G}$) contains the global rank of a given configurations, obtained by averaging the ranks on individual benchmarks. The second-to-last column (Rank$_\mathrm{L}$) presents the ranks obtained by each method within the group of configurations that share the same value of factorization rank r. The last row shows success rates obtained by ordinary GP, where fitness is calculated from all tests.

To provide an aggregated perspective on the impact of factorization rank r on SFIMX performance, we average the ranks of configurations that share the same value of factorization rank r, leading to the following ranking:

SFIMX-F	SFIMX-H	SFIMX-L
14.10	16.44	18.41

Table 12.4: Success rate (×100) of best-of-run individuals, averaged over 30 evolutionary runs. SFIMX configurations have population size increased by the factor of $1/\alpha$. Bold marks the best result for each benchmark.

α	Cmp6	Cmp8	Maj6	Maj8	Mux6	Par5	Dsc1	Dsc2	Dsc3	Dsc4	Dsc5	Mal1	Mal2	Mal3	Mal4	Mal5	Rank_L	Rank_G
									SFIMX-F									
0.9	56	3	66	0	**100**	6	10	6	46	0	3	76	50	90	16	76	7.34	21.29
0.8	63	0	83	0	**100**	**16**	6	0	60	0	20	76	66	93	33	**100**	6.16	17.44
0.7	73	**16**	73	0	**100**	0	10	10	43	0	10	70	66	93	20	90	6.34	19.15
0.6	66	3	**86**	3	**100**	6	3	6	70	0	10	76	86	96	70	90	5.38	14.85
0.5	70	13	73	**6**	**100**	0	13	0	80	0	13	93	70	96	50	**100**	5.03	13.59
0.4	66	10	73	0	**100**	6	16	6	86	0	30	**100**	90	96	53	**100**	4.38	11.41
0.3	73	10	80	0	**100**	0	**43**	**56**	86	0	26	86	90	**100**	60	**100**	3.88	11.24
0.2	90	10	83	0	**100**	0	23	13	93	**6**	**50**	**100**	86	**100**	73	**100**	3.09	8.53
0.1	**93**	3	73	0	**100**	0	33	70	90	**6**	23	90	**96**	**100**	**76**	**100**	3.41	9.44
									SFIMX-H									
0.9	66	3	66	0	**100**	0	13	3	30	0	3	83	30	93	23	96	6.78	21.47
0.8	70	3	70	**6**	**100**	3	3	13	53	0	10	83	50	76	23	96	5.75	18.18
0.7	86	6	80	3	**100**	0	0	0	50	0	6	96	90	93	30	**100**	4.78	15.91
0.6	73	0	70	0	**100**	0	3	0	63	0	13	96	73	93	30	**100**	5.53	17.97
0.5	73	0	66	**6**	**100**	0	0	10	66	0	20	86	80	96	43	96	5.28	16.91
0.4	86	0	73	0	**100**	0	0	10	70	0	33	83	83	96	60	**100**	4.56	15.68
0.3	70	0	76	0	**100**	0	6	26	53	0	16	90	**96**	93	60	**100**	4.50	14.94
0.2	80	3	66	0	**100**	0	33	26	**96**	0	26	86	93	93	60	**100**	3.88	12.76
0.1	86	0	20	0	**100**	0	23	23	70	0	26	86	93	**100**	63	**100**	3.94	14.18
									SFIMX-L									
0.9	56	3	60	0	**100**	3	0	0	53	0	16	90	26	86	13	96	5.97	21.74
0.8	43	6	**86**	0	**100**	3	3	6	26	0	6	86	43	73	3	96	6.00	21.26
0.7	66	6	56	3	**100**	3	0	3	46	0	3	90	50	80	23	96	5.62	19.88
0.6	66	6	56	0	**100**	0	0	10	66	0	13	90	40	83	16	86	5.88	21.32
0.5	76	13	80	**6**	**100**	3	6	10	70	0	16	90	73	83	33	**100**	3.28	12.38
0.4	93	13	80	0	**100**	0	3	16	63	6	26	83	80	93	60	**100**	3.19	12.44
0.3	93	10	83	0	**100**	3	10	10	86	0	33	90	93	86	50	96	3.00	11.91
0.2	76	3	73	0	**100**	0	3	16	56	0	20	63	63	80	40	**100**	4.97	18.06
0.1	6	0	0	0	**100**	0	0	0	6	0	0	70	70	76	20	**100**	7.09	26.68
									RSS									
0.9	46	0	50	0	**100**	0	0	0	30	0	0	70	53	56	10	90	5.56	27.21
0.8	33	0	53	0	96	0	0	0	26	0	0	63	33	80	13	96	5.91	28.32
0.7	13	0	40	3	**100**	0	0	0	26	0	0	66	33	63	13	90	6.06	27.44
0.6	43	0	56	0	96	0	0	0	26	0	0	70	23	60	3	**100**	5.68	27.65
0.5	23	0	63	0	**100**	0	0	3	43	0	3	80	43	66	16	93	4.06	24.88
0.4	33	0	56	0	**100**	0	0	0	33	0	0	83	46	80	16	96	4.32	25.03
0.3	56	0	60	0	**100**	0	0	0	43	0	3	83	56	80	23	96	3.71	24.38
0.2	36	0	50	0	**100**	0	0	0	23	0	0	73	66	93	16	**100**	4.76	25.26
0.1	20	0	26	0	**100**	0	0	0	36	0	0	76	60	80	16	**100**	4.94	25.94
									GP									
	54	2	54	0	98	2	0	0	44	0	12	88	0	68	0	88		26.26

The ranking suggests that higher values of r contributes positively to the overall performance of the method. Possible explanation for such a behavior is that NMF employed by SFIMX is working with more features, allowing it to build a more complex, and possibly more accurate model of interaction outcomes in G.

The global ranking of configurations confirms the above observation and reveals that the best configuration is SFIMX-F in combination with $\alpha = 0.2$. It achieves the average rank of 8.53, outperforming all control methods by a significant margin. The next three places in the ranking are also occupied by SFIMX-F, albeit working with α equal to 0.1, 0.3 and 0.4, respectively. The gains in success rates for lower α are also present in the two remaining groups that share the same value of r. SFIMX-H achieves the best results with $\alpha = 0.2$, followed by $\alpha = 0.1$ and $\alpha = 0.3$.

Table 12.5: Post-hoc analysis of Friedman's test conduced on ranks achieved by the best performing configurations from Table 12.4. Significant values ($\alpha = 0.05$) are marked in bold.

	SFIMX-$F_{0.2}$	SFIMX-$L_{0.3}$	SFIMX-$H_{0.2}$	GP	$RSS_{0.3}$
SFIMX-$F_{0.2}$		0.866	0.808	**0.000**	**0.000**
SFIMX-$L_{0.3}$			1.000	**0.003**	**0.007**
SFIMX-$H_{0.2}$				**0.005**	**0.011**
GP					
$RSS_{0.3}$				0.999	

SFIMX-L, on the other hand, performs best for α ranging between 0.3 and 0.5. For the values of α lower than 0.3, its performance however deteriorates, to the point that for $\alpha = 0.1$ it is the only SFIMX configuration that ranks behind GP and a few RSS setups. Nevertheless, SFIMX-L still delivers decent performance and for its preferred setting $\alpha = 0.3$ it surpasses GP, RSS and SFIMX-H on most benchmarks, taking 5th place in the global ranking. We find this result remarkable given that SFIMX-L uses roughly an order of magnitude fewer weights (in W) and features (in H) than SFIMX-F and SFIMX-H. This corroborates our hypothesis that the interaction outcomes in G are significantly correlated and lend themselves to high compression without affecting the overall performance of the method. Good performance of SFIMX-L is particularly appealing, as low r implies low computational overhead of factorization: for SFIMX-L, it amounts only to roughly 6 percent of the total cost of calculating the $1,000|T|$ program-test interactions.

The success rates of SFIMX for individual benchmarks are always the best among the considered methods – see the values marked in bold in Table 12.4. For SFIMX-F, the SFIMX variant that overall fares the best, there is at least one setting of α that makes SFIMX succeed systematically on three problems (Mux6, Mal1, and Mal5), i.e., in every run (success rate 100). In that respect, it is not equaled by any control method.

SFIMX performs also well in qualitative terms. It manages to produce solutions for all problems, while GP never solves Maj8, Dsc1, Dsc2, Dsc4, Mal2 and Mal4, and RSS never solves Cmp8, Par5, Dsc1 and Dsc4, and hardly ever solves Maj8, Dsc2 and Dsc5. On those hard problems, SFIMX is in most cases remarkably resistant to the setting of α: for Cmp8 and Dsc1, it succeeds for most values of α in the range $[0.1, 0.9]$, and for Dsc5 for $\alpha \in [0.1, 1.0]$. The only exceptions are Maj8 and Dsc4 where it manages to solve the problem only for some α, and no more than twice in 30 runs.

To statistically evaluate these results, we employed the Friedman's test for multiple achievements of multiple subjects, for all configurations presented in the table (i.e., in relation to global average ranks in the rightmost column in Table 12.4). The obtained p-value is 4.36×10^{-9}, which strongly indicates that at least one method performs significantly different from the remaining ones. We conducted the post-hoc analysis using symmetry test [135], which revealed that SFIMX-F configurations employing $\alpha \in [0.1, 0.4]$ are significantly better than GP and all RSS configurations. Notably, similar observation holds also for SFIMX-H with $\alpha = 0.2$ and for SFIMX-L with $\alpha \in [0.3, 0.5]$. The differences are insignificant within the groups that share the same α and between the corresponding SFIMX configurations for different values of r.

For additional insight, we also rank and apply the Friedman's test to the best performing configurations in each group

SFIMX-$F_{0.2}$	SFIMX-$L_{0.3}$	SFIMX-$H_{0.2}$	$RSS_{0.3}$	GP
1.875	2.375	2.438	4.094	4.219

Table 12.6: Success rate (×100) of best-of-run individuals, averaged over 30 evolutionary runs. SFIMX configurations have their runtime (number of generations) increased by the factor of $1/\alpha$. Bold marks the best result for each benchmark.

α	Cmp6	Cmp8	Maj6	Maj8	Mux6	Par5	Dsc1	Dsc2	Dsc3	Dsc4	Dsc5	Mal1	Mal2	Mal3	Mal4	Mal5	Rank_L	Rank_G
									SFIMX-F									
0.9	53	6	73	0	**100**	6	0	10	30	0	13	76	26	83	6	90	6.25	19.62
0.8	70	16	56	0	**100**	6	3	6	43	0	0	60	60	96	36	96	5.41	16.44
0.7	66	10	66	6	**100**	6	0	3	36	0	10	66	46	76	20	93	6.16	17.38
0.6	70	20	76	6	**100**	0	0	0	43	0	6	66	56	93	50	93	5.38	16.47
0.5	76	3	60	0	**100**	0	0	23	50	0	10	76	50	76	30	76	6.09	19.31
0.4	73	3	70	10	**100**	6	3	**30**	53	0	0	80	46	96	43	86	4.59	13.94
0.3	56	13	53	0	96	0	13	10	73	0	16	83	63	**100**	56	96	4.62	13.91
0.2	83	30	66	0	**100**	3	**20**	26	83	0	20	70	76	**100**	60	93	3.19	9.38
0.1	**86**	26	**100**	0	**100**	0	10	13	**96**	0	16	86	90	93	30	**100**	3.31	9.31
									SFIMX-H									
0.9	50	3	66	0	**100**	3	0	0	40	0	6	60	26	90	10	96	6.66	21.72
0.8	63	6	66	0	**100**	**13**	0	6	53	0	10	70	40	63	20	96	5.72	17.81
0.7	60	3	63	0	**100**	10	0	16	53	0	6	63	63	90	56	86	6.00	17.38
0.6	63	33	83	0	**100**	3	0	3	70	0	16	76	**100**	**100**	46	96	3.84	11.66
0.5	83	6	63	0	**100**	0	6	23	70	0	6	80	73	86	43	96	4.78	13.78
0.4	76	6	80	3	**100**	3	6	26	56	0	20	56	70	80	60	96	4.19	12.03
0.3	50	10	66	0	**100**	0	3	10	66	0	3	70	63	86	40	86	6.03	17.97
0.2	83	13	50	0	**100**	0	13	23	73	0	0	60	76	96	60	96	4.41	14.34
0.1	**86**	10	70	0	**100**	0	10	26	73	0	16	70	76	**100**	**66**	90	3.38	11.38
									SFIMX-L									
0.9	50	0	53	3	**100**	3	0	0	40	0	6	66	43	86	0	73	6.59	23.53
0.8	43	0	56	3	**100**	3	0	0	43	0	3	**90**	30	56	0	86	6.56	23.59
0.7	56	3	73	0	**100**	6	0	3	66	0	6	73	36	86	6	90	5.31	19.09
0.6	76	30	66	6	**100**	0	0	3	66	0	**36**	80	63	80	20	96	4.28	14.00
0.5	60	**36**	73	0	**100**	0	3	3	60	0	16	83	83	86	10	90	4.41	15.06
0.4	76	6	80	10	**100**	10	3	10	73	0	3	70	60	93	33	96	3.41	11.84
0.3	**86**	33	86	3	**100**	3	3	10	76	0	20	70	76	96	53	93	2.75	9.06
0.2	83	23	73	**16**	**100**	0	6	20	66	0	6	50	50	73	23	80	4.47	16.19
0.1	0	16	0	0	**100**	0	0	0	20	0	0	56	30	86	16	76	7.22	26.84
									RSS									
0.9	30	0	26	0	96	0	0	0	33	0	0	70	30	73	6	86	5.34	28.53
0.8	56	0	30	0	**100**	0	0	0	6	0	0	80	36	66	6	73	5.16	26.88
0.7	30	0	36	0	93	0	0	0	20	0	0	63	36	63	13	83	5.47	29.16
0.6	46	0	33	0	**100**	0	0	0	30	0	0	70	33	53	3	83	5.25	27.84
0.5	46	0	30	0	93	0	0	0	23	0	0	80	33	60	6	100	5.16	26.59
0.4	46	0	26	0	93	0	0	0	23	0	0	56	30	66	13	80	5.88	29.78
0.3	50	0	50	0	93	0	0	0	16	0	0	60	20	73	16	90	5.06	28.38
0.2	63	3	56	0	**100**	0	0	0	13	0	0	60	43	86	10	93	3.88	24.09
0.1	36	6	53	0	**100**	0	0	0	26	0	0	43	43	90	16	93	3.81	24.47
									GP									
	54	2	54	0	98	2	0	0	44	0	12	88	0	68	0	88		24.25

Table 12.5 presents the p-values for the hypothesis that a setup in a row is better than a setup in a column (the significant p-values are marked in bold). This comparison confirms the observations made earlier that the performance improvement of SFIMX configurations relative to control methods GP and RSS is significant.

12.4.4 Results for increased runtime

Increasing population size by the factor of $1/\alpha$ is one viable method of spending the evaluation cycles spared thanks to SFIMX. The other interesting possibility, which we experimentally validate in the following, is to let SFIMX runs last longer with the same population size. Non-SFIMX methods are thus allowed to perform up to 200 generations of search, while SFIMX increases this

Table 12.7: Post-hoc analysis of Friedman's test conduced on ranks achieved by the best performing configurations from Table 12.6. Significant values ($\alpha = 0.05$) are marked in bold.

	SFIMX-$L_{0.3}$	SFIMX-$F_{0.1}$	SFIMX-$H_{0.1}$	$RSS_{0.3}$	GP
SFIMX-$L_{0.3}$		1.000	0.891	**0.000**	**0.003**
SFIMX-$F_{0.1}$			0.949	**0.000**	**0.006**
SFIMX-$H_{0.1}$				**0.004**	0.056
$RSS_{0.3}$					
GP				0.913	

number accordingly (i.e., $1/\alpha$ times), up to 2000 generations for $\alpha = 0.1$. Let us again emphasize that, as a result of this action, the computational budget, measured as the number of performed interactions, is the same for SFIMX and control methods.

Table 12.6 reports the success rates resulting from 30 runs of each configuration on every benchmark. The table is formatted in the same manner as in the previous experiment. The results are largely consistent with those obtained for an increased population size (cf. Table 12.4). We observe close-to-systematic improvement in success rates for low values of α. This trend holds for all benchmark problems and regardless of the factorization rank r. The average ranks of configurations sharing the same value of r are as follows:

SFIMX-F	SFIMX-H	SFIMX-L
15.08	15.34	17.69

which suggests that SFIMX-F tends to achieve the highest success rates, though the differences are less prominent than in the previous section.

The global ranking of configurations, shown in the last column of Table 12.6, reveals that the overall best result is achieved by SFIMX-$L_{0.3}$ with the rank of 9.06. We find this result particularly interesting, given that SFIMX-L is also the most computationally efficient variant of SFIMX. Notably, the very same configuration performed the best in the SFIMX-L group in combination with an increased population size (cf. Table 12.4), which suggests that $\alpha = 0.3$ is the optimal choice for SFIMX-L. The subsequent places in the ranking are taken by SFIMX-$F_{0.1}$, SFIMX-$F_{0.2}$ and SFIMX-$H_{0.1}$ with the ranks 9.31, 9.38 and 9.38, respectively. The best results are thus consistently obtained for $\alpha \in [0.1, 0.3]$, though there appears to be more variation in ranks for higher α, and between the different settings of r. All SFIMX configurations except for SFIMX-$L_{0.1}$ rank before any control configuration, which only strengthens our claim that the computational budget spared by SFIMX can be invested elsewhere to boost its search performance. GP and RSS occupy the last few places in the ranking, achieving the lowest success rates of all compared methods. Compared to the variant with increased population size, RSS performs slightly worse, surpassing GP only when $\alpha = 0.2$.

In terms of statistical significance, we perform Friedman test in relation to global average rank in Table 12.6. The test is conclusive (p-value is 1.42×10^{-6}). Post-hoc analysis shows that indeed the top-ranking configurations (SFIMX-$L_{0.3}$, SFIMX-$F_{0.1}$, SFIMX-$F_{0.2}$) are significantly better than GP and all RSS configurations.

For an additional insight, we also rank and apply the Friedman's test to the best performing configuration in each group, which leads to the following ranking:

SFIMX-$L_{0.3}$	SFIMX-$F_{0.1}$	SFIMX-$H_{0.1}$	$RSS_{0.3}$	GP
2.062	2.156	2.531	4.219	4.344

Table 12.7 presents the p-values for the hypothesis that a setup in a row is better than a setup in a column (the significant p-values are marked in bold). This comparison confirms the observations

made earlier that the performance improvement of the best SFIMX configurations relative to control methods GP and RSS is significant.

By juxtaposing the success rates in Table 12.6 with those in Table 12.4, we arrive at the conclusion that it pays-off more to invest the spare computational budget in large populations rather than letting candidate solutions evolve longer. Though the differences between the corresponding configurations in both tables are not statistically significant, the trend is clearly in favor of the former.

12.5 Adaptive test selection in SFIMX

In this section, we consider alternative means of drawing the tests to interact with in line 3 of Algorithm 8. Our aim is to improve SFIMX's performance by replacing the uniform probability distribution that is used there and changing it adaptively as a run progresses. The motivation is that the outcomes of interactions with certain tests are likely to be more difficult to predict than others. Should that be true, then it might be particularly beneficial to compute these interactions rather than use SFIMX to estimate them from the remaining elements of G. And conversely: if there are grounds to claim that, for some candidate solutions and tests, the outcomes of their interactions are easy to predict, then it may be worth not to perform them and let them be estimated by the NMF.

12.5.1 Methods

Recall from Section 7.2 that test difficulty meant as the probability of t being passed can be estimated based on the historical interaction outcomes, gathered for instance throughout an evolutionary run. This is the key idea behind the extensions of SFIMX we describe in the following: the interaction matrix is inspected in each generation, certain statistics are updated and used to parameterize the probability distribution employed by the SAMPLE function in line 3 of Algorithm 8. Crucially, only the actual (computed) interactions from the historical Gs are used for this purpose – the estimated ones in $\hat{G}$s are ignored. Therefore, the prediction of test difficulty is based on the interaction outcomes that were computed in the past, so no extra interactions are required. By inspecting the interaction matrix in every generation, the methods adapt the probability distribution to the current state of population, shifting SAMPLE's attention towards the tests that were the most challenging in the recent generations.

We propose three methods of controlling SAMPLE. The first one, dubbed **DIFF**, is based on *test difficulty* $p(t)$, which we define as the number of candidate solutions that did not pass test t up to the current point of evolutionary run (i.e., up to the previous generation). This indicator is updated in each generation based on the current population S, where the initial population starts with a vector p initially zeroed for every test in T:

$$p(t) \leftarrow p(t) + |s \in S : \ g(s,t) = 1|. \tag{12.5.1}$$

In evaluation with SFIMX, p is L_1-normalized (i.e., divided by $\sum_t p(t)$) and used as the probability distribution by SAMPLE. As a consequence, the more difficult tests have a higher chance of being included in T'. We expect this probability distribution to improve the estimation of interaction outcomes, as the outcomes of interactions with easier tests should be in principle also easier to predict. More difficult tests, on the other hand, are typically characterized by greater variability in the interaction outcomes, thus we expect SFIMX to commit greater errors in their estimation.

In the second variant, **DIST**, we employ the concept of distinctions (Eq. 9.4.1), borrowed from competitive coevolution [86]. By analogy to p above, we use the current population S to update the total number of distinctions $q(t)$ made so far by t in the following way:

$$q(t) \leftarrow q(t) + |\{(s_1, s_2) \in S \times S : g(s_1, t) \neq g(s_2, t)\}|, \tag{12.5.2}$$

where q is initially all-zeroes. Large values of q indicate the tests that *inform* the search algorithm about the differences between candidate solutions, rather than about their absolute performance. As in DIFF, q is normalized and used by Sample. Interestingly, there seems to be a link between the concept of distinctions and the concept of error variance used in works that combine GP with coevolution [317].

While the above two methods use ad-hoc measures of tests difficulty to infer which interaction outcomes are more difficult to predict, there exists a more direct alternative: measuring the difficulty of prediction using the difference between the known elements of G and the corresponding estimated elements of $\hat{G}$. The resulting method, referred to as **ERR** in the following, accumulates those errors $e(t)$ throughout the run similarly to DIST and DIFF:

$$e(t) \leftarrow e(t) + \sum_{s \in S} |g(s, t) - \hat{g}(s, t)|, \tag{12.5.3}$$

where e is then normalized and used in Sample. In contrast to p (Eq. 12.5.1) and q (Eq. 12.5.2) that depend solely on the performance of candidate solutions, e depends also on the quality of factorization conducted by the NMF, and so reflects the inherent estimation difficulty (which depends, among others, on the factorization rank r).

Let us emphasize that, even though the probability distributions defined above replace the uniform distribution used in the basic SFIMX, drawing of the tests to conduct interactions with is still performed independently for each candidate solution (and with replacement between individual candidates). It is thus possible (though unlikely for the common proportions of population size and number of tests) for some tests to be absent from T' (randomly drawn subset of tests in SFIMX, $T' \subset T$) in a given generation. Also, let us reiterate that all these methods accumulate only the outcomes of the interactions that have been actually performed in the past, so that the quantities aggregated by the above formulas are not biased (at least directly) by the errors committed by NMF when estimating the unknown interaction outcomes.

12.5.2 Experimental setup

In the following experiment, we are interested in verifying whether the extensions DIFF, DIST, and ERR improve the performance of a regular SFIMX. For this reason, we consider as the baselines the configurations introduced in Section 12.4.1:

- **SFIMX-F** that uses the highest factorization rank $r = \min(m, n)$,
- **SFIMX-H** with $r = n/2$, and
- **SFIMX-L** that uses the smallest rank $r = \lceil \log_2 n \rceil$.

See Section 12.4.1 for more details regarding the above methods. Spare evaluation cycles are invested in larger population size (cf. Section 12.4.3). We limit our attention to $\alpha \in \{0.1, 0.2, 0.3\}$, as these proved most effective for the original SFIMX. We follow the experimental settings and benchmark problems used in Section 12.4.

Table 12.8: Success rate (×100) of best-of-run individuals, averaged over 30 evolutionary runs. The second-to-last column (Rank_L) presents the ranks within the groups of configurations sharing r. The last column contains the global rank (Rank_G) of a given configuration.

α	Cmp6	Cmp8	Maj6	Maj8	Mux6	Par5	Dsc1	Dsc2	Dsc3	Dsc4	Dsc5	Mal1	Mal2	Mal3	Mal4	Mal5	Rank_L	Rank_G
								SFIMX-F										
0.1	93	3	73	0	100	0	33	70	90	6	23	90	96	100	76	100	7.50	16.34
0.2	90	10	83	0	100	0	23	13	93	6	50	100	86	100	73	100	7.56	16.78
0.3	73	10	80	0	100	0	43	56	86	0	26	86	90	100	60	100	8.62	19.19
								DIFF										
0.1	100	96	96	0	100	0	56	90	100	13	50	100	100	100	83	100	3.94	8.22
0.2	100	96	100	0	100	0	56	83	100	30	56	100	93	100	83	100	3.94	8.16
0.3	100	76	100	0	100	3	60	86	100	10	76	93	100	100	80	100	3.59	7.16
								DIST										
0.1	100	23	100	0	100	0	36	60	100	6	30	100	93	100	70	100	5.62	11.66
0.2	100	36	100	0	100	0	30	43	86	0	43	93	93	100	60	96	7.22	15.41
0.3	80	16	90	0	100	3	16	6	93	0	10	80	90	100	40	100	8.81	20.62
								ERRS										
0.1	96	23	93	0	100	0	53	66	100	3	53	90	93	100	83	100	5.62	11.69
0.2	93	10	100	0	100	0	36	36	93	0	46	90	100	100	56	100	7.16	15.41
0.3	93	13	90	0	100	3	20	40	96	0	43	86	66	93	46	100	8.41	18.50
								SFIMX-H										
0.1	86	0	20	0	100	0	23	23	70	0	26	86	93	100	63	100	7.69	21.06
0.2	80	3	66	0	100	0	33	26	96	0	26	86	93	93	60	100	7.31	20.38
0.3	70	0	76	0	100	0	6	26	53	0	16	90	96	93	60	100	8.19	22.66
								DIFF										
0.1	100	80	83	0	100	0	36	70	96	3	43	86	96	100	80	100	3.84	12.25
0.2	100	83	100	0	100	0	46	56	100	0	50	86	86	100	73	100	4.16	12.59
0.3	100	80	100	0	100	0	63	60	100	10	60	93	100	96	66	100	3.31	9.91
								DIST										
0.1	96	26	86	0	100	0	26	36	93	0	26	86	90	96	46	100	6.44	18.44
0.2	96	13	100	0	100	0	23	16	90	0	33	70	76	100	40	100	7.22	19.62
0.3	93	23	96	0	100	0	6	26	96	0	40	70	66	90	16	100	7.78	21.53
								ERRS										
0.1	90	0	63	0	100	0	30	26	96	3	26	80	90	96	73	96	7.31	20.72
0.2	96	20	80	0	100	0	13	40	73	0	30	76	86	96	40	100	7.38	21.16
0.3	80	6	80	0	100	0	3	30	83	0	30	80	76	100	73	100	7.38	21.41
								SFIMX-L										
0.1	6	0	0	0	100	0	0	0	6	0	0	70	70	76	20	100	9.12	29.38
0.2	76	3	73	0	100	0	3	16	56	0	20	63	63	80	40	100	7.34	26.81
0.3	93	10	83	0	100	3	10	10	86	0	33	90	93	86	50	96	5.62	21.09
								DIFF										
0.1	40	86	3	0	100	0	0	0	30	0	0	83	86	90	53	100	6.69	24.78
0.2	100	90	100	0	100	0	36	40	100	13	63	63	80	86	43	100	3.88	14.69
0.3	100	56	100	0	100	0	20	76	100	10	66	100	100	100	80	100	2.72	9.44
								DIST										
0.1	20	23	0	0	100	0	0	0	13	0	0	60	46	86	23	93	9.31	29.75
0.2	96	43	100	0	100	0	13	16	90	0	23	53	53	63	30	100	6.47	22.78
0.3	90	20	93	0	100	0	16	6	93	0	13	76	76	96	40	100	5.91	22.56
								ERRS										
0.1	20	23	0	0	100	0	0	6	16	0	0	60	70	90	33	100	8.00	27.44
0.2	93	16	96	0	100	0	0	3	93	0	23	53	53	76	46	93	7.66	25.69
0.3	93	23	96	0	100	3	13	13	83	0	30	70	50	93	60	100	5.28	20.75

12.5.3 Success rates

Table 12.8 reports the success rates obtained by particular configurations on each benchmark, grouped by the value of the factorization rank r, starting from SFIMX-F in the upper part of the table, SFIMX-H in the middle, and SFIMX-L at the bottom. In each group, we show the results obtained by the three proposed SFIMX extensions and the baseline SFIMX for $\alpha \in \{0.1, 0.2, 0.3\}$. The second-to-last column presents the ranks obtained by each configuration within the group.

Table 12.9: Post-hoc analysis of Friedman's test conduced on ranks achieved by the SFIMX-F configurations from the upper part of Table 12.4. Significant values ($\alpha = 0.05$) are marked in bold.

	$\alpha = 0.3$			
	SFIMX-F	DIFF-F	DIST-F	ERRS-F
SFIMX-F			0.999	
DIFF-F	**0.001**		**0.000**	**0.004**
DIST-F				
ERRS-F	0.963		0.917	

We first analyze the behavior of the proposed extensions in groups of configurations that share the same value of r. By comparing the extended SFIMX variants with the corresponding baseline SFIMX-F configurations, we observe close to systematic improvement in success rates in the methods that employ DIFF. This observation holds regardless of α, albeit the differences are particularly prominent for $\alpha = 0.3$ (the average rank 3.59 vs. 8.62). It is encouraging the see large improvements in success rates even for the hardest problems in our benchmark suite, including Cmp8, Par5, Dsc1, Dsc4 and Dsc5. The two remaining extensions, DIST and ERRS, also tend to rank before their corresponding baseline SFIMX-F configurations, the only exception being DIST for $\alpha = 0.3$ (the average rank 8.81 vs. 8.62). The differences are however not as evident as in the case of DIFF. For instance, when $\alpha = 0.2$ the average ranks of DIST and ERRS are 7.22 and 7.16, respectively, which is only a minor improvement over the baseline configuration with the rank of 7.56. Of these two extensions, it is rather difficult to clearly indicate which leads to better results; for $\alpha = 0.1$ they even obtain the same rank of 5.62.

To investigate the statistical significance, we conduct Friedman's test for multiple achievements of multiple subjects [159]. We repeat the test three times for different values of α, each time obtaining the p-value that is $\ll 0.001$, which strongly indicates that at least one method performs significantly different from the remaining ones. The post-hoc analysis using the symmetry test [135] (also conducted thrice) indicates that the improvement of DIFF relative to the regular SFIMX-F across all values of α is indeed significant. For $\alpha = 0.2$ and $\alpha = 0.3$, DIFF is also significantly better than DIST and ERRS. For brevity, in Table 12.9 we show only the outcomes for $\alpha = 0.3$.

Similar observations can be made for SFIMX-H and SFIMX-L. When DIFF is used, the improvements are unquestionable. The configurations that combine SFIMX with DIFF achieve top ranks, and we observe systematically higher success rates for all considered values of α. In terms of statistical significance, Friedman test performed independently in each group and for each α is conclusive ($p \ll 0.001$). Post-hoc analysis, omitted here for brevity, consistently shows that DIFF's rank is significantly better than SFIMX's and than the two alternative extensions (DIST and ERRS).

For a more detailed insight, we also rank all 36 configurations on each benchmark and present the averaged ranks in the last-but-one column of Table 12.8. The best overall average rank of 7.16 is achieved by DIFF-$F_{0.3}$. The first non-DIFF methods in the ranking are DIST-$F_{0.1}$ and ERR-$F_{0.1}$, occupying 6th (rank 11.66) and 7th place (rank 11.69) in the ranking, respectively. The first baseline configuration in the ranking is SFIMX-$F_{0.1}$ with the 13th place and the rank of 16.34. The ranking reveals also that the methods employing the DIST sampling typically rank better than ERR, which may suggest that in certain cases distinctions are more useful than the errors from factorization for shaping the probability distribution of test drawing. We are particularly pleased to see that configurations based on SFIMX-L behave so well (e.g. DIFF-$F_{0.3}$ with the 5th place and the rank of 9.44), regardless of the choice of parameters. By using the smallest factorization rank $r = \lceil \log_2 n \rceil$, they are on average computationally cheaper than the other methods (cf. Table 12.3).

Table 12.10: Post-hoc analysis of Friedman's test for DIFF. The p-value 1.54×10^{-9}. Significant values (0.05) are in bold.

	SFIMX $\alpha = 0.3$					
	DIFF-F	SFIMX-F	DIFF-H	SFIMX-H	DIFF-L	SFIMX-L
DIFF-F		**0.002**	0.941	**0.000**	0.981	**0.000**
SFIMX-F				0.908		0.998
DIFF-H		**0.040**		**0.001**		**0.010**
SFIMX-H						
DIFF-L		**0.021**	1.000	**0.000**		**0.005**
SFIMX-L				0.991		

This advantage is crucial for larger problems that involve more tests and yield larger interaction matrices.

Figure 12.1 shows the average best-of-generation fitness graphs for particular methods and benchmark problems, with 95% confidence intervals marked as semi-transparent bands. We present only the best performing configurations, i.e those that used $\alpha = 0.3$ and DIFF extension. By comparing the extended SFIMX variants with the corresponding baseline configurations (marked by the same color on the graph), we observe improvements in learning speed and best-of-generation fitness in the methods that employ DIFF. For certain problems, including Cmp8, Dsc4, Dsc5, the differences between the methods are particularly prominent, showing clear superiority of DIFF to the original SFIMX. The best performance is achieved either by DIFF-F or DIFF-L, depending on the problem. DIFF-H performs slightly worse or falls in between the already mentioned configurations, but most importantly, it still achieves lower fitness than any baseline setup.

To provide an aggregated perspective on the performance of the DIFF extension against the other methods, we employ the Friedman's test again. Post-hoc analysis using symmetry test [135] is shown Table 12.10. We report the results only for $\alpha = 0.3$ that consistently leads to the highest success rates (cf. Table 12.8). The comparison indicates that the improvement of DIFF relative to the regular SFIMX across all configurations is indeed significant. In particular, DIFF-F is significantly better than SFIMX-F, SFIMX-H and SFIMX-L. Similar observations can be made for DIFF-H and DIFF-L, which outperform their counterparts. For other values of α (0.1 and 0.2), DIFF still delivers statistically significant improvements and surpasses SFIMX on most benchmarks (cf. Table 12.8).

12.5.4 Visualization of measures of test difficulty

In order to gain a deeper insight into the differences between DIFF, DIST and ERR, in Fig. 12.2 we plot the changes of the normalized difficulty of all 32 tests in the Par5 problem, one of the harder benchmarks in our suite, for the first 100 generations. To create the graphs, we first computed the test difficulty according to Eqs. 12.5.1-12.5.3 (Section 12.5.1). Then, the resulting 32-element vectors were averaged across 50 runs to form the mean test difficulty. Finally, the resulting 32×100 matrix was normalized and presented as a heatmap, with brighter colors corresponding to harder tests.

Judging from the graphs in Fig. 12.2, DIFF starts to differentiate tests' difficulty from the early stages of evolutionary runs and manages to maintain that differentiation with time. As evolution proceeds, the brighter stripes fade away, some faster than others, as candidate solutions in the population adapt and solve the more difficult tests. DIST behaves similarly, however the differences in the difficulty measure seem to be more subtle and, judging from somewhat lower performance of this variant, insufficient to shape the probability distribution in a way that would make a significant

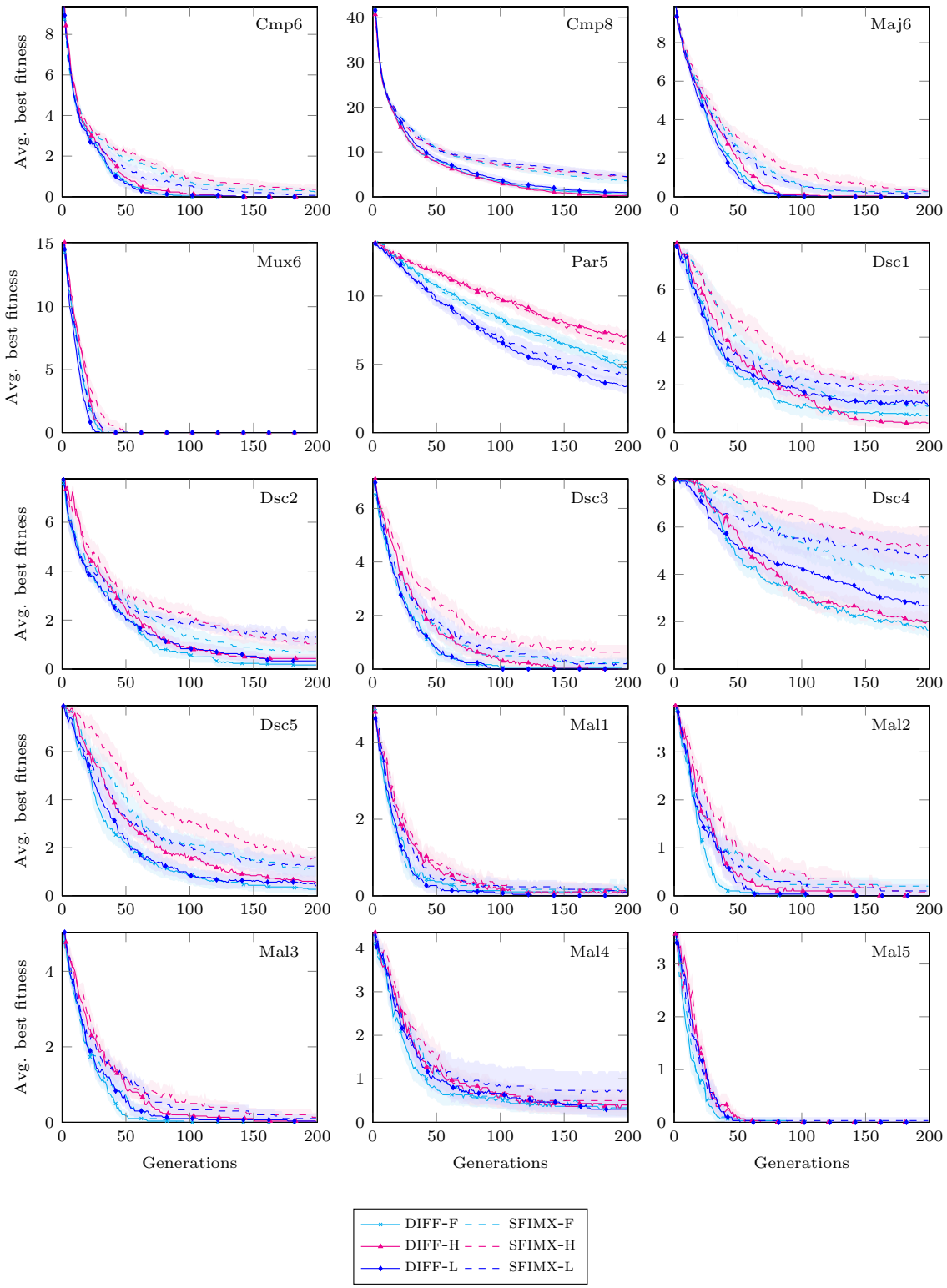


Figure 12.1: Average and .95-confidence interval of the best-of-generation fitness for DIFF and the baseline SFIMX configurations when $\alpha = 0.3$.

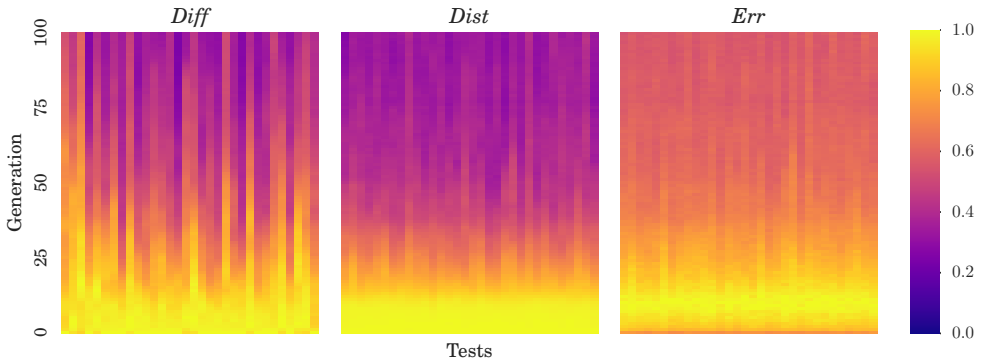


Figure 12.2: Normalized difficulty of 32 tests in Par5 problem during the first 100 generations as indicated by the three methods: DIFF, DIST and ERR. The brighter the color, the more difficult the test.

Table 12.11: Mean absolute error (×100) when reconstructing G in SFIMX, averaged over 30 evolutionary runs. Green cells indicate configurations with lower error than the regular SFIMX.

Method	Cmp6	Cmp8	Maj6	Mux6	Par5	Dsc1	Dsc2	Dsc3	Dsc4	Dsc5	Mal1	Mal2	Mal3	Mal4	Mal5
SFIMX															
F	11.83	9.75	12.73	14.21	19.93	11.39	16.45	17.14	9.77	16.66	26.72	26.05	26.21	26.03	28.52
H	13.31	10.76	14.13	16.28	22.09	12.23	16.75	18.13	10.48	16.44	30.66	29.49	27.34	28.36	32.69
L	13.53	13.01	14.63	17.47	21.31	9.80	14.23	17.43	8.69	13.66	28.74	25.97	24.80	24.76	31.61
DIFF															
Full	11.66	9.50	12.53	14.20	20.17	10.74	15.10	15.92	10.35	14.84	26.62	24.95	24.18	23.07	29.32
Half	12.96	10.47	14.05	15.60	21.57	12.41	16.73	17.39	10.36	14.45	29.95	27.49	26.10	25.69	33.49
Log	13.40	12.83	14.38	16.80	20.50	9.22	13.96	15.80	8.90	12.82	27.88	24.71	24.81	23.23	30.66
DIST															
F	11.66	9.61	12.82	14.52	20.38	12.06	16.13	16.55	10.09	15.00	27.25	25.94	25.01	24.31	29.96
H	13.19	10.57	14.16	16.04	21.64	11.33	16.84	17.89	11.33	16.30	30.80	29.76	26.75	27.71	32.76
L	13.49	12.92	14.34	17.03	21.20	10.19	15.98	16.81	8.53	15.64	29.85	26.20	24.90	24.12	31.69
ERR															
F	11.79	9.47	12.31	14.33	21.51	19.25	19.15	17.93	15.30	21.86	28.14	28.44	26.35	26.35	30.24
H	13.22	10.42	14.12	16.24	23.45	20.87	20.87	19.83	13.50	23.66	32.39	32.15	29.01	30.25	34.61
L	14.07	12.82	14.84	17.62	23.58	14.42	19.42	18.47	10.65	19.40	31.52	31.04	27.02	29.30	33.67

impact on the success rate. ERR is characterized by the most uniform distribution of all three methods. Apparently, NMF's estimation error turns out to be roughly uniformly distributed across all tests. All in all, the visualization provided in Fig. 12.2 confirms that neither DIST nor ERR discern the tests well enough to guide search more effectively than the corresponding SFIMX baselines, and this is most likely the reason why they are unable to enhance SFIMX's performance.

Finally, we verify whether the methods improve the predictive capabilities of SFIMX. In Table 12.11 we present the mean absolute estimation error calculated as $\sum_i |G_i - \hat{G}_i|$, where i iterates over generations of a run ($i \in [1, 200]$), for the same Par5 problem. To calculate this error, we compute, alongside with the estimated interaction matrix $\hat{G}$, also the complete interaction matrix G, even though the latter is never used by fitness. Green color indicates that an extended configuration achieves lower error than its SFIMX counterpart. DIFF and DIST tend to systematically decrease the error on most of problems, however the reduction typically does not exceed 12%. ERR performs noticeably worse, managing to improve the error on just a handful of problems. By juxtaposing these results with the success rates from Table 12.8, we may conclude that the performance improvement of DIFF originates, at least to some extent, from the more accurate predictions.

12.5.5 Summary

The SFIMX extensions proposed in this section corroborate our claims that tests not only vary in difficulty, but also that this variability can be exploited to make search more effective. We

used this property of test-based problems to shape the probability with which the tests are being drawn for interactions. That proved overall beneficial, without incurring significant computational overheads. The best variant, DIFF, proved significantly better than its counterpart, regardless of the choice of parameters. Though the boost in success rate offered by DIST and ERR with respect to SFIMX is rather minor, the trend is clear.

It is not unlikely that further improvements could be achieved with introduction of additional mechanisms. For instance, all methods considered here base their estimates on the entire history of evolutionary run, i.e. on the interaction outcomes for candidate solutions from the most recent generations, as well as of the not so well-performing candidates from the initial generations. One may argue that some form of *aging* applied to the estimates (e.g., exponential smoothing) may make them more up-to-date, better tuned to the capabilities of the candidate solutions in the current population, and thus more beneficial for success rate.

A natural follow-up of the experiments performed here could be engagement of DIFF, DIST, ERR, and techniques alike to other methods that derive search objectives form interaction matrices. DOF is the obvious candidate here as it employs NMF to obtain a multi-dimensional characterization of candidate solutions that could directly benefit from such an enhancement (cf. Section 10.2).

12.6 Automatic tunning of α in SFIMX

The most important parameter of SFIMX is α that controls the number of interactions to be computed. In essence, it trades-off evaluation precision for computational performance. By setting α to a low value, we may expect significant speed-up in terms of learning speed, however at the expense of more noisy fitness. It thus makes more sense to use low α early on to maximize its potential. Conversely, high values of α imply more accurate evaluation and may be argued to be particularly useful towards the end of evolution, when precise evaluation is required to differentiate candidate solutions and make further progress. Clearly, both arguments are feasible, which was in part confirmed by the overall observation made in the experiments in this chapter, i.e. that $\alpha \in [0.4, 0.61]$ typically leads to best performance of SFIMX (cf. Table 12.1). Therefore, rather than committing to a single value of α, it may be more beneficial to automatically adjust it to the transient characteristics of search process. By increasing α online, we hope to improve the evaluation accuracy and provide the search process with better guidance.

12.6.1 ADASFIMX

In this section, we propose an extensions of SFIMX dubbed ADASFIMX that automatically adapts α based on the current learning performance. The method is shown in Algorithm 9. The basic idea to start with low α and gradually increase it as the search performance starts to deteriorate. To this end, we determine the rate at which α grows by *exploration step* $\gamma \in (0, 1)$ and discretize α into $1/\gamma$ *levels* and start with $level = 1$ (corresponding to $\alpha = \gamma$). In line 7 of Algorithm 9, we compute α based on the current level and γ. The adaptation mechanism is inspired by the algorithms designed for multi-armed bandit problems [11]: with probability $1 - \epsilon$, we use the *current* level of α, and with probability ϵ, we perform *exploration* on the next level (i.e., with greater α; lines 5-6 of Algorithm 9). Once the level is chosen, we proceed as in regular SFIMX, computing fitness from estimated interaction matrix $\hat{G}$ as in (12.2.1).

For the current and the next level, we track the change in learning speed defined as:

$$learn\_speed = \frac{\Delta p}{\Delta e},$$

Algorithm 9 ADASFIMX - adaptive tunning of α based on the statistics of learning performance.

Require: window size w, exploration rate ϵ, exploration step γ

1: $level \leftarrow 1$ ▷ A global variable
2: $\mathcal{H} \leftarrow \{\}$ ▷ History of learning speed

3: **function** AdaSfimx(S, T)
4: $curr \leftarrow level$
5: **if** Rand$(0, 1) < \epsilon$ **then**
6: $curr \leftarrow level + 1$
7: $\alpha \leftarrow curr \times \gamma$
8: $F \leftarrow$ SFIMX(S, T, α)
9: UpdateLevel$(curr, \alpha)$
10: **return** F
11:
12: **function** UpdateLevel($curr$, α)
13: $learn\_speed = \frac{\Delta p}{\Delta e}$
14: $\mathcal{H}[curr] \leftarrow$ Concat$(\mathcal{H}[curr], learn_speed)$
15: $V_{level} \leftarrow w$ most recent values from $\mathcal{H}[level]$
16: $V_{level+1} \leftarrow w$ most recent values from $\mathcal{H}[level + 1]$
17: **if** $len(V_{level}) < w$ **or** $len(V_{level+1}) < w$ **then**
18: **return**
19: **if** mean$(V_{level}) <$ mean$(V_{level+1})$ **then** ▷ switch condition
20: **if** $\alpha < 1$ **then**
21: $level \leftarrow level + 1$
22: $\mathcal{H}[level] \leftarrow \varnothing$

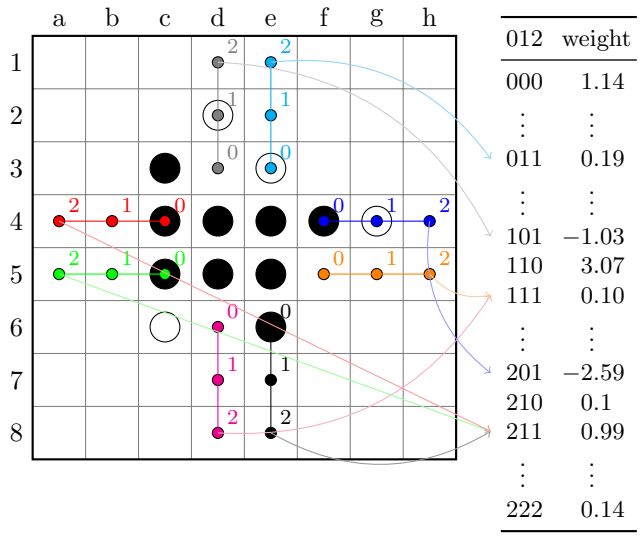
where Δp is the the most recent increase in the objective performance (see Section 12.6.3), i.e. the difference of the objective performance of the best individual from the current generation and the best-of-generation individual from the previous generation, and Δe denotes the computational effort (the total number of interactions) that caused this increase.

By observing the learning speeds for two consecutive levels, the algorithm eventually makes a decision to switch from *level* to *level* $+1$ (lines 19-22 in Algorithm 9). Such a change is permanent and increases the fitness precision, at the expense of the number of interactions to compute. Notice that in order to make the decision based on the most recent data, we take into account only w most recent learning speeds (lines 15-16 of Algorithm 9). We also make sure that the algorithm gathers enough samples prior to making any decisions regarding the level of α (line 17 of Algorithm 9).

12.6.2 Position evaluation in Othello with n-tuple networks

In this section, we describe the problem of learning position evaluation function in the game of Othello (cf. Section 5.4.1), which will be later used in the experiment to assess the effectiveness of ADASFIMX.

Due to an extreme number of possible board states in Othello, the position (board) evaluation function cannot be learned directly and has to be approximated. Among a range of formalisms that can be used to that aim, Bledsoe and Browning [24] introduced *n-tuple networks*, a particularly powerful and computationally efficient function approximators, which were originally applied to the problem of optical character recognition. In the context of games, they were first used by Buro [41], and later popularized by Lucas [227]. The main advantages of n-tuple networks are their conceptual simplicity and efficiency in realizing nonlinear mappings, in which they surpass the weighted piece counter (WPC), a simple linear combination of board states, often used as a baseline method for implementing position evaluation functions.



012	weight
000	1.14
⋮	⋮
011	0.19
⋮	⋮
101	−1.03
110	3.07
111	0.10
⋮	⋮
201	−2.59
210	0.1
211	0.99
⋮	⋮
222	0.14

Figure 12.3: A straight 3-tuple employed eight times for the given board position (symmetric sampling). The eight symmetric expansions of the 3-tuple return $0.19 - 1.03 + 2 \times 0.1 - 2.59 + 3 \times 0.99 = -0.26$.

An n-tuple network consists of m tuples of up to n different board locations each. For a given board state $\mathbf{b}$, the network outputs the sum of values returned by the individual tuples, each of which depends on the occupation of indicated board locations indicated. The ith tuple, where $i = 1, \ldots, m$, contains a sequence of n_i board locations $(loc_{ij})_{j=1,\ldots,n_i}$, $n_i \leq n$, and an associated look-up table LUT_i. The table contains weights for each possible combination of states that can be constructed using the locations in the tuple. The value of an n-tuple network can be thus interpreted as a position evaluation function f:

$$\begin{aligned} f(\mathbf{b}) = \sum_{i=1}^{m} f_i(\mathbf{b}) &= \sum_{i=1}^{m} \mathrm{LUT}_i \left[idx\left(\mathbf{b}_{loc_{i1}}, \ldots, \mathbf{b}_{loc_{in_i}}\right)\right] \\ idx\left(\mathbf{v}\right) &= \sum_{j=1}^{|\mathbf{v}|} v_j c^{j-1}, \end{aligned}$$

where $\mathbf{b}_{loc_{ij}}$ is a board value at location loc_{ij}, $\mathbf{v}$ is a sequence of board values ($0 \leq v_k < c$, for $k = 1, \ldots, |\mathbf{v}|$), and c denotes the number of possible board values ($c = 3$ for Othello). As a result, one look-up table of length n_i contains 3^{n_i} weights.

To improve the effectiveness of n-tuple networks, we also employ *symmetric sampling* that exploits the inherent symmetries of a game board [227]. In symmetric sampling, a single tuple is employed 8 times, and returns one value for each possible board rotation and reflection (see Fig. 12.3).

Board inversion

A game-playing agent (candidate solution) selects its moves by unrolling the game tree from the current state (by one or more levels) and applying the position evaluation function to them, preferring the moves that lead to positions of the highest value. Since we expect the agent to play both black and white, and given that Othello is an almost symmetric game, we train only the black agent, and when playing as the white player we temporarily flip all the pieces on the board in order to interpret the board from the black player's perspective. Then we select the best move according to the position evaluation function, flip the pieces back, and play the white piece

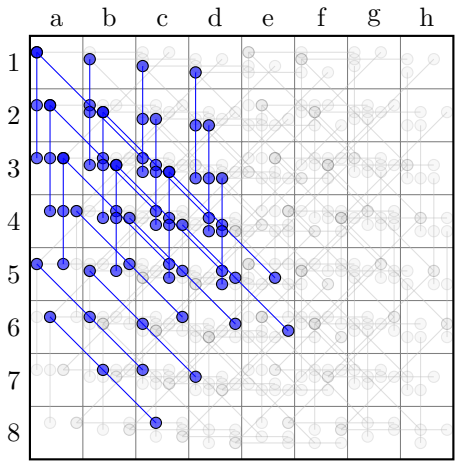


Figure 12.4: All 24 straight 3-tuples (648 weights). Their symmetric expansions have been shown in light gray.

in the selected location. This method is called *board inversion* [236, 306] and has been found more efficient than using doubled function or output negation [142].

Systematic n-tuple networks

In order choose the shapes of tuples, Lucas [227] proposed to randomly generate a small number of long snake-shaped sequences. However, it has been recently shown that a large number of short systematically selected tuples leads to better results [142]. The *systematic n-tuple network* consists of all possible vertical, horizontal, and diagonal n-tuples of the same length (see Fig. 12.4). Its smallest representative is a network of 1-tuples. Thanks to symmetric sampling, only 10 of such 1-tuples are required to cover an 8×8 Othello board, and a 10×1-tuple network contains $10\times3^1 = 30$ weights. The comparison of different n-tuple architectures has been performed by Jaśkowski and Szubert [146]. The authors report that the combination of straight 4-tuples and (square) 2×2 tuples achieves the best results.

12.6.3 Experimental setup

In the following experiments, we evaluate SFIMX and ADASFIMX on the problem of position evaluation in the game of Othello. To this end, we employ Coevolutionary CMA-ES, a variant of one-population CoEA that combines the covariance matrix adaptation evolutionary strategy [120], a state-of-the-art continuous black-box optimization method, with a competitive fitness function. The evolving candidate solutions (game strategies) are evaluated by playing a double game (cf. Section 5.4.1) against each other in a round-robin fashion (cf. Section 3). The fitness is the sum of scores obtained in these games, where in a single game the win, loss or draw counts as 1, 0.5 or 0 points, respectively. The elements of interaction matrix G are in the $[0, 1]$ range and correspond to the outcome of a double game.

We strictly follow the experimental setup from [146] in order to directly compare the results. Candidate solutions are represented as real-valued vectors interpreted as the weights of a systematic n-tuple network consisting of all straight 4-tuples and all square 2×2-tuples. The step-size σ of CMA-ES is initialized to 1. The initial starting point for CMA-ES is generated by sampling the weights uniformly from the range $[-0.1, 0.1]$. We use the population size $\lambda = 400$ that was found to achieve the best results in [146]. All algorithms were run 5 times.

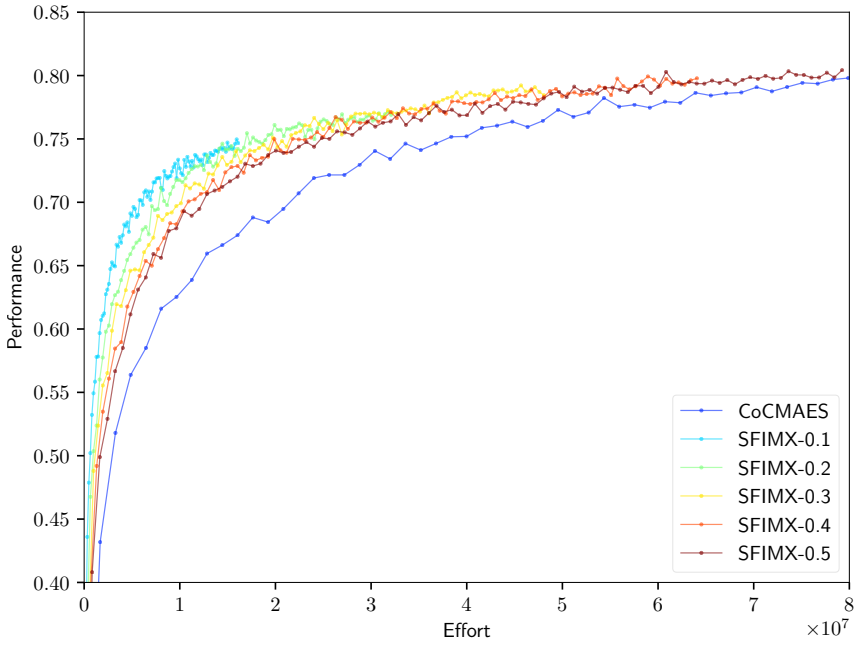


Figure 12.5: Average performance of the best-of-generation individuals obtained using SFIMX with constant α. Each run was stopped after 1000 generations.

To objectively measure the progress of algorithms, we employed an external performance measure consisting in playing (double) Othello games against 11 previously published position evaluation functions on 1000 opening positions. In that aspect, we follow the protocol described in [146], to which we refer for more details. Every 10 generations, we report the average of 22000 games.

The ADASFIMX setup, which automatically adapts α during a run (cf. Section 12.6.1), employs the same objective performance measure for computing the learning speed but estimates it only from 100 positions to reduce the computational overhead. Both SFIMX and ADASFIMX use factorization rank $r = \log(\lambda) \approx 10$.

12.6.4 Experimental verification

In the first experiment, we were interested in verifying whether basic SFIMX is a viable method for accelerating the vanilla round-robin-based CoCMAES. To this aim, we control the fraction of interactions to be calculated by the parameter $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ in Algorithm 8. Since the number of interactions is reduced by a factor of $1 - \alpha$, in each generation we spare $(1 - \alpha)|S|^2$ interactions.

Figure 12.5 plots the objective performance of the best-of-generation individual as a function of computational effort (the number of interactions). The results clearly demonstrate that SFIMX provides significant speed-up while maintaining the overall performance. The curve that represents the baseline CoCMAES is dominated by the other methods for most of the time, indicating that the same performance can be achieved much faster. For instance, SFIMX with $\alpha = 0.1$ achieves the performance level of 0.75 nearly 2.2 times quicker than the baseline CoCMAES. Despite the fact that we stopped the run after 1000 generations, it can already be observed that it would not achieve the same performance level as the baseline method, since the higher the performance levels, the more precise fitness is required.

The observations made above led us to the design of ADASFIMX that automatically adjusts α during evolution (cf. Section 12.6.1 and Algorithm 9). ADASFIMX uses step size $\gamma = 0.1$, leading to 10 levels of α $\{0.1, 0.2, \ldots, 1.0\}$. We also set the window size $w = 20$ so that the mean learning

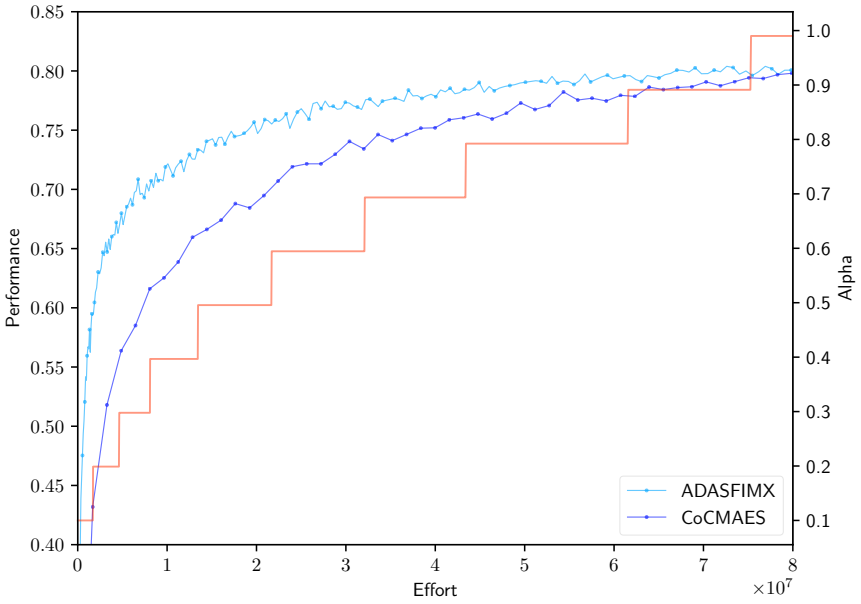


Figure 12.6: Average performance of the best-of-generation individuals obtained using CoCMAES and CoCMAES augmented by ADASFIMX.

speed is computed from the 20 most recent learning speeds in $\mathcal{H}$. Exploration rate ϵ is set to 0.1. Preliminary experiments (not reported here for brevity) revealed that ADASFIMX behaves roughly the same for $w \in \{10, 20, 50\}$.

In Fig. 12.6, we compare the objective performance of the best-of-generation individuals obtained using ADASFIMX and the baseline CoCMAES. We also plot α to visualize how it changes during the learning. The results demonstrate that ADASFIMX learns faster throughout the evolutionary run. CoCMAES achieves a similar level of performance, but does so at a much slower pace. For instance, the performance of roughly 0.8 (i.e., 80% probability of winning against the reference players) is achieved by ADASFIMX after 6×10^7 games, i.e., about a quarter faster than by CoCMAES. This clearly indicates that ADASFIMX manages to achieve state-of-the-art performance at significantly lower computational expense.

The fact that both methods ultimately achieve the same level of performance is not surprising since ADASFIMX eventually converges to the round-robin CoCMAES. As shown by the plot, ADASFIMX starts computing the complete interaction matrix G at the end of evolution, shortly after reaching the milestone of 7.5×10^7 interactions. From that point onward, it progresses just as a regular CoCMAES, achieving roughly the same performance.

12.7 Chapter summary

Surrogate Fitness via Factorization of Interaction Matrix aims at reducing the overall computational cost of fitness evaluation, while benefiting from the specifics of test-based problems. Given the outcomes of a random sample of all solution-test interactions, SFIMX uses non-negative matrix factorization to predict the outcomes of remaining interactions and so estimate the fitness of candidate solutions. In an empirical evaluation, we demonstrated its effectiveness as a surrogate model to speed up the evaluation process (Section 12.4.2). When performance is of the highest priority, it is possible to invest the computational budget spared by SFIMX into a larger population (Section 12.4.3), or increased evolution time (Section 12.4.4). In either case, we observed significant improvement in the probability of finding the ideal candidate solution. We also extended SFIMX

by replacing the uniform probability distribution used to draw tests to interact with during evaluation with the distribution biased towards more difficult tests (Section 12.5). Finally, we showed the method for an automatic choice of α that controls the overall number of interactions to be computed during evaluation (Section 12.6).The conceptual contribution that is worth emphasizing is SFIMX's abstraction from domain specifics: interaction outcomes are predicted from other interaction outcomes, and the method is agnostic about the internals of function evaluation. We find this advantage pivotal in comparison to most of traditional surrogate models, which predict fitness from solution's characteristics, and are thus domain-specific.

Chapter 13

Conclusions

13.1 Summary

Our experience with test-based problems covered in this thesis, suggests that interaction matrices in test-based problems form a rich source of multi-faceted characterization of candidate solutions and tests. This insight, along with the observed tendency to treat evaluation functions as a black box, shaped the design of the proposed framework for discovery of search objectives. In the following, we summarize its key features and related claims we made in this thesis:

1. Search objectives reflect only selected characteristics of candidate solutions. Rather than providing objective assessment of candidate solution's quality, their role is to guide search by creating a useful gradient towards better performing solutions.

2. Search objectives may help avoiding overfocusing on some tests and diversify the population by continuously shifting the selection pressure to different objectives. The emphasis is thus put on different aspects of candidate solution's quality, thereby promoting behaviorally diverse candidate solutions with the potential to perform well, and minimizing the risk of premature convergence.

3. Search objectives are derived in every generation of evolutionary run independently. In this way, they change dynamically to capture new and interesting behaviors that emerge in interaction matrices during evolution.

4. Discovery of search objectives is a heuristic process. The dominance relation induced by these objectives is not required to be consistent with the original relation defined on tests. It is, however, capable of providing search gradient that is strong enough to efficiently solve a range of problems of practical interest. Heuristic approach allows us also to minimize to computational cost associated with it, and enables the objectives to be derived online during evolution.

5. Discovery of search objectives turns a single-objective problem into a multi-objective one. This allows us to avoid pitfalls of scalar evaluation discussed in Chapter 6, and enhances evolvability, i.e. the possibility to progressively find better solutions. Multi-objective approach also facilitates exploration of different trade-offs between possibly conflicting search objectives.

6. The framework for discovery of search objectives is independent of any particular test-based problem and, in particular, of representation of candidate solutions and tests. The only requirement is the access to an interaction matrix.

As argued elsewhere [187, 160], we postulate that treating fitness function as a black box is unjustified, especially when more detailed information on solution's characteristics, like interaction outcomes, is easily available. Such information typically requires more effort in conceptual analysis, implementation and computational expense to harness, but, as we demonstrated in this thesis, these costs may pay off in the long run with a more effective search method.

The algorithms for discovery of search objectives proposed here may make significant impact in many application areas of EC that comply with the test-based paradigm, like design of complex artifacts, synthesis of controllers, and learning game strategies — in short, everywhere where the quality of a candidate solution can be determined only by evaluating it against a number of tests.

In a broader perspective, the results presented in this dissertation form yet another argument for the quest for alternative means of driving search in heuristic algorithms [183, 194]. In comparison to the research on hyper-heuristics [40], where the question is *how* to perform search, in our framework we focus on *what* to drive the search with. Indeed, in many domains there are neither conceptual nor technical obstacles for distilling more precise and useful information from candidate solutions. To that aim, we employed here the interaction matrix, but potential other approaches abound. Given the potential benefits evidenced in this dissertation, such opportunities should be exploited more often in research and practice of test-based problems, and we strongly encourage the readers to consider this path.

13.2 Contributions

The main contributions of this thesis may be summarized as follows:

- Identification of pitfalls accompanying scalar evaluation in test-based problems, including the phenomenon of evaluation bottleneck that originates in the aggregation of outcomes of interactions between candidate solutions and tests. [Chapter 6]
- The concept of performance profile, a generic and domain-independent tool for multi-criteria evaluation of solutions produced by evolutionary algorithms solving test-based problems. [Chapter 7]
- Introduction and formalization of the unified framework for automatic discovery of search objectives in test-based problems, intended to widen the evaluation bottleneck by providing search algorithms with richer information on solutions' characteristics. [Chapter 8]
- The algorithm for discovery of search objectives by heuristic clustering of outcomes of interactions between candidate solutions and tests (DOC). [Chapter 9]
- The algorithm for discovery of search objectives by non-negative factorization of interaction matrix (DOF). [Chapter 10]
- The universal approach for transforming interaction matrices generated in continuous domains to a form that is appropriate for the proposed framework for automatic discovery of search objectives in test-based problems. [Chapter 11]
- The algorithm for reducing the overall computational cost of evaluation in test-based problems that learns to predict outcomes of interactions taking place between candidate solutions and tests (SFIMX). [Chapter 12]
- The extension of SFIMX algorithm that assesses the per-test estimation errors by comparing the predicted outcomes with the actual ones, and use these errors to bias the sampling

of interactions that are conducted, i.e. making the interactions on the 'hard' tests to be executed rather than predicted. [Chapter 12]

- The extension of SFIMX algorithm that automatically adapts its parameters. [Chapter 12]
- Experimental evaluation of the proposed algorithms on selected test-based problems. [Chapters 9 – 12]
- Implementation of the proposed algorithms as a common software framework written in Java and Python programming languages.

13.3 Future work

The work presented in this thesis may be extended in many directions. Let us point out a few of them in the following list:

- In practical terms, the proposed framework for discovery of search objectives broadens the evaluation bottleneck in information flow between an evaluation function and a selection operator. Concerning other possibilities, a particularly interesting area of future work is to provide *search operators* with a more detailed information on behavior of candidate solutions and thus making their actions more directed and responsive to the current state of evolution. For instance, based on a factorization of interaction matrix (cf. Chapter 10), a crossover operator could first select behaviorally-dissimilar parents (or even parents with complementary behavior), and then fuse the capabilities elaborated by particular individuals.
- Discovered search objectives are transient in their nature, meaning that they cannot *accumulate* knowledge about the characteristics of individual tests along an evolutionary search process. One one hand, starting from scratch, as a *tabula rasa*, is desired as it allows search objectives to adapt to the current stage of evolution. On the other, we find it worth investigating whether it is possible to benefit from the knowledge acquired in the previous generations, while remaining also responsive to the ongoing changes in the population via continuous learning.
- We find it worthwhile to briefly discuss possible extensions of the algorithms devised in this thesis. In particular, DOC and DOF can discover search objectives only if their existence is manifested *behaviorally*, i.e. reflected in the outcomes of interactions between candidate solutions and tests. If such behavioral patterns do not manifest themselves in interaction outcomes, there is no grounds to discover them. It is therefore interesting to ask whether we could mine for patterns (not necessarily behavioral, as we did in this thesis) in e.g. *representation* of candidate solutions.
- Search objectives derived by the proposed algorithms dwell in the linear space. Though the experimental evidence presented in this thesis suggests that in case of many test-based problems this is not a limitation, it may be expected that complex dependencies (behavioral patterns) between interaction outcomes arise from applying candidate solutions to various tests, and such interactions may need a more complex, *nonlinear* model to be well captured. This is particularly true for SFIMX, where outcomes of interaction are predicted based on linear combinations of the elements in the matrices W and H, resulting from non-negative factorization. These observations point to the natural follow-up research, in which nonlinear models of interaction outcomes could be explored. Neural networks operating in auto-associative regime such as autoencoders [133] that implement a 'bottleneck' architecture seem to be particularly adequate to this task.

Naturally, the above ideas are not even close to fully exhausting the scope of possible future research directions. Prospectively, we hope to see the extensions of the proposed framework for discovery of search objectives to other paradigms of learning, including for instance reinforcement learning. This could open the door to tackling even more difficult problems, including those of 'uncompromising' nature [126, 129]. Only time will bring the answers to this and other suppositions formulated in this thesis.

Bibliography

[1] The On-line Encyclopedia of Integer Sequences, Published Electronically at https://oeis.org, 2017.

[2] Thomas Ackling, Bradley Alexander, and Ian Grunert. Evolving Patches for Software Repair. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1427–1434. ACM, 2011.

[3] Eva Alfaro-Cid, Juan J Merelo, F Fernández de Vega, Anna Isabel Esparcia-Alcázar, and Ken Sharman. Bloat Control Operators and Diversity in Genetic Programming: A Comparative Study. *Evolutionary Computation*, 18(2):305–332, 2010.

[4] Louis Victor Allis et al. *Searching for Solutions in Games and Artificial Intelligence*. Rijksuniversiteit Limburg, 1994.

[5] Peter J Angeline. An Alternate Interpretation of the Iterated Prisoner's Dilemma and the Evolution of Non-mutual Cooperation. In *Proceedings 4th Artificial Life Conference*, pages 353–358, 1994.

[6] Peter J. Angeline. Subtree Crossover: Building Block Engine or macromutation? In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 9–17, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann. URL http://ncra.ucd.ie/COMP41190/SubtreeXoverBuildingBlockorMacromutation_angeline_gp97.ps.

[7] Peter J. Angeline and Jordan B. Pollack. Competitive Environments Evolve Better Solutions for Complex Tasks. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pages 264–270, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann. ISBN 1-55860-299-2. URL http://www.demo.cs.brandeis.edu/papers/icga5.pdf.

[8] Andrea Arcuri and Xin Yao. A Novel Co-evolutionary Approach to Automatic Software Bug Fixing. In Jun Wang, editor, *2008 IEEE World Congress on Computational Intelligence*, pages 162–168, Hong Kong, 1-6 June 2008. IEEE Computational Intelligence Society, IEEE Press. doi: 10.1109/CEC.2008.4630793.

[9] Daniel Ashlock and Colin Lee. Agent-case Embeddings for the Analysis of Evolved Systems. *IEEE Transactions on Evolutionary Computation*, 17(2):227–240, 2013.

[10] Wirt Atmar. Notes on the Simulation of Evolution. *IEEE Transactions on Neural Networks*, 5(1): 130–147, 1994.

[11] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine learning*, 47(2-3):235–256, 2002.

[12] Robert Axelrod. *Genetic Algorithms and Simulated Annealing*, chapter The Evolution of Strategies in the Iterated Prisoner's Dilemma, pages 32–41. Morgan Kaufman, Los Altos, CA, 1987.

[13] Robert Axelrod. The Evolution of Strategies in the Iterated Prisoner's Dilemma. *The dynamics of norms*, pages 199–220, 1987.

[14] Yaniv Azaria and Moshe Sipper. Gp-gammon: Genetically Programming Backgammon Players. *Genetic Programming and Evolvable Machines*, 6(3):283–300, 2005.

[15] Thomas Back. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford university press, 1996.

[16] Thomas Back, Ulrich Hammel, and H-P Schwefel. Evolutionary Computation: Comments on the History and Current State. *IEEE transactions on Evolutionary Computation*, 1(1):3–17, 1997.

[17] Roland Badeau, Nancy Bertin, and Emmanuel Vincent. Stability Analysis of Multiplicative Update Algorithms and Application to Nonnegative Matrix Factorization. *IEEE Transactions on Neural Networks*, 21(12):1869–1881, 2010.

[18] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming – an Introduction; on the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann, San Francisco, CA, USA, January 1998. ISBN 1-55860-510-X. URL http://www.elsevier.com/wps/find/bookdescription.cws_home/677869/description#description.

[19] Lawrence Beadle and Colin Johnson. Semantically Driven Crossover in Genetic Programming. In Jun Wang, editor, *Proceedings of the IEEE World Congress on Computational Intelligence*, pages 111–116, Hong Kong, 1-6 June 2008. IEEE Computational Intelligence Society, IEEE Press. doi: 10.1109/CEC.2008.4630784. URL http://results.ref.ac.uk/Submissions/Output/1423275.

[20] Patrick Berarducci, Demetrius Jordan, David Martin, and Jennifer Seitzer. GEVOSH: Using Grammatical Evolution to Generate hashing functions. In R. Poli, S. Cagnoni, M. Keijzer, E. Costa, F. Pereira, G. Raidl, S. C. Upton, D. Goldberg, H. Lipson, E. de Jong, J. Koza, H. Suzuki, H. Sawai, I. Parmee, M. Pelikan, K. Sastry, D. Thierens, W. Stolzmann, P. L. Lanzi, S. W. Wilson, M. O'Neill, C. Ryan, T. Yu, J. F. Miller, I. Garibay, G. Holifield, A. S. Wu, T. Riopka, M. M. Meysenburg, A. W. Wright, N. Richter, J. H. Moore, M. D. Ritchie, L. Davis, R. Roy, and M. Jakiela, editors, *GECCO 2004 Workshop Proceedings*, Seattle, Washington, USA, 26-30 June 2004. URL http://www.cs.bham.ac.uk/~wbl/biblio/gecco2004/WUGW001.pdf.

[21] Michael W Berry, Murray Browne, Amy N Langville, V Paul Pauca, and Robert J Plemmons. Algorithms and Applications for Approximate Nonnegative Matrix Factorization. *Computational statistics & data analysis*, 52(1):155–173, 2007.

[22] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution Strategies–a Comprehensive Introduction. *Natural computing*, 1(1):3–52, 2002.

[23] Alan D. Blair and Jordan B. Pollack. What makes a good co-evolutionary learning environment. *Australian Journal of Intelligent Information Processing Systems*, 4(3/4):166–175, 1997.

[24] Woodrow Wilson Bledsoe and Iben Browning. Pattern Recognition and Reading by Machine. In *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, pages 225–232. ACM, 1959.

[25] Stefan Bleuler, Johannes Bader, and Eckart Zitzler. Reducing Bloat in GP with Multiple Objectives. In Joshua Knowles, David Corne, and Kalyanmoy Deb, editors, *Multiobjective Problem Solving from Nature: from concepts to applications*, Natural Computing, chapter 9, pages 177–200. Springer, 2008. doi: 10.1007/978-3-540-72964-8_9.

[26] Josh Bongard. Behavior Chaining-incremental Behavior Integration for Evolutionary Robotics. In *ALIFE*, pages 64–71, 2008.

[27] Josh C Bongard and Hod Lipson. Nonlinear System Identification Using Coevolution of Models and Tests. *Evolutionary Computation, IEEE Transactions on*, 9(4):361–384, 2005.

[28] Olivier Bousquet and Léon Bottou. The Tradeoffs of Large Scale Learning. In *Advances in neural information processing systems*, pages 161–168, 2008.

[29] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge university press, 2004.

[30] Jeremy S. Bradbury and Kevin Jalbert. Automatic Repair of Concurrency Bugs. In Massimiliano Di Penta, Simon Poulding, Lionel Briand, and John Clark, editors, *Proceedings of the 2nd International Symposium on Search Based Software Engineering (SSBSE '10)*, Benevento, Italy, 7-9 September 2010. URL http://www.ssbse.org/2010/fastabstracts/ssbse2010_fastabstract_04.pdf. Fast abstract.

[31] Markus Brameier and Wolfgang Banzhaf. *Linear Genetic Programming*. Number XVI in Genetic and Evolutionary Computation. Springer, 2007. ISBN 0-387-31029-0. URL http://www.springer.com/west/home/default?SGWID=4-40356-22-173660820-0.

[32] Leo Breiman. Bagging Predictors. *Machine learning*, 24(2):123–140, 1996.

[33] Dimo Brockhoff and Eckart Zitzler. Are All Objectives Necessary? on Dimensionality Reduction in Evolutionary Multiobjective Optimization. In *Parallel Problem Solving from Nature-PPSN IX*, pages 533–542. Springer, 2006.

[34] Sébastien Bubeck et al. Convex Optimization: Algorithms and Complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[35] A. Bucci, J. Pollack, and E. De Jong. Automated Extraction of Problem Structure. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 501–512. Springer, 2004.

[36] Anthony Bucci. *Emergent Geometric Organization and Informative Dimensions in Coevolutionary Algorithms*. PhD thesis, Waltham, MA, USA, 2007.

[37] Anthony Bucci and Jordan B Pollack. Order-theoretic Analysis of Coevolution Problems: Coevolutionary Statics. In *Proceedings of the GECCO-2002 Workshop on Coevolution: Understanding Coevolution*, pages 229–235, 2002.

[38] Richard L Burden and J Douglas Faires. Numerical Analysis. 2001. *Brooks/Cole, USA*, 2001.

[39] RL Burden and JD Faires. Numerical Analysis, Cengage Learning, 2010. Technical report, ISBN 978-0-538-73351-9, 1989.

[40] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An Emerging Direction in Modern Search Technology. In *Handbook of metaheuristics*, pages 457–474. Springer, 2003.

[41] Michael Buro. An Evaluation Function for Othello Based on Statistics. *Technical Report 31, NEC Research Institute*, 1997.

[42] Xian-Bin Cao, Hong Qiao, and John Keane. A Low-cost Pedestrian-detection System with a Single Optical Camera. *IEEE Transactions on Intelligent Transportation Systems*, 9(1):58–67, 2008.

[43] John Cartlidge and Seth Bullock. Combating Coevolutionary Disengagement by Reducing Parasite Virulence. *Evolutionary Computation*, 12(2):193–222, 2004.

[44] J.P. Cartlidge. *Rules of Engagement: Competitive Coevolutionary Dynamics in Computational Systems*. PhD thesis, University of Leeds, 2004.

[45] Tsung-Han Chan, Wing-Kin Ma, Chong-Yung Chi, and Yue Wang. A Convex Analysis Framework for Blind Separation of Non-negative Sources. *IEEE Transactions on Signal Processing*, 56(10): 5120–5134, 2008.

[46] Kumar Chellapilla and David B Fogel. Evolving Neural Networks to Play Checkers without Relying on Expert Knowledge. *IEEE Transactions on Neural Networks*, 10(6):1382–1391, 1999.

[47] Kumar Chellapilla and David B. Fogel. Evolving an Expert Checkers Playing Program without Using Human Expertise. *IEEE Transactions on Evolutionary Computation*, 5(4):422–428, 2001.

[48] Eric C Chi and Tamara G Kolda. On Tensors, Sparsity, and Nonnegative Factorizations. *SIAM Journal on Matrix Analysis and Applications*, 33(4):1272–1299, 2012.

[49] Siang Y Chong, Mei K Tan, and Jonathon David White. Observing the Evolution of Neural Networks Learning to Play the Game of Othello. *IEEE Transactions on Evolutionary Computation*, 9(3):240–251, 2005.

[50] Siang Yew Chong, Peter Tino, and Xin Yao. Relationship between Generalization and Diversity in Coevolutionary Learning. *IEEE Transactions on computational intelligence and AI in games*, 1(3): 214–232, 2009.

[51] S.Y. Chong and X. Yao. Behavioral Diversity, Choices and Noise in the Iterated Prisoner's Dilemma. *Evolutionary Computation, IEEE Transactions on*, 9(6):540–551, 2005.

[52] S.Y. Chong, P. Tiňo, and X. Yao. Measuring Generalization Performance in Coevolutionary Learning. *Evolutionary Computation, IEEE Transactions on*, 12(4):479–505, 2008.

[53] S.Y. Chong, P. Tiňo, D.C. Ku, and X. Yao. Improving Generalization Performance in Co-evolutionary Learning. *Evolutionary Computation, IEEE Transactions on*, (99):1–1, 2012.

[54] David M Clark. Evolution of Algebraic Terms 1: Term to Term Operation Continuity. *International Journal of Algebra and Computation*, 23(05):1175–1205, 2013.

[55] Dave Cliff and Geoffrey F Miller. Tracking the Red Queen: Measurements of Adaptive Progress in Co-evolutionary Simulations. In *European Conference on Artificial Life*, pages 200–218. Springer Berlin Heidelberg, 1995.

[56] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. *Evolutionary Algorithms for Solving Multi-objective Problems*, volume 5. Springer, 2007.

[57] Robert J Collins and David Jefferson. *Antfarm: Towards Simulated Evolution*. Computer Science Department, University of California, 1990.

[58] Alison Cozad, Nikolaos V Sahinidis, and David C Miller. Learning Surrogate Models for Simulation-based Optimization. *AIChE Journal*, 60(6):2211–2227, 2014.

[59] Nichael Lynn Cramer. A Representation for the Adaptive Generation of Simple Sequential Programs. In John J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and the Applications*, pages 183–187, Carnegie-Mellon University, Pittsburgh, PA, USA, 24-26 July 1985. URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/icga1985/icga85_cramer.pdf.

[60] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and Exploitation in Evolutionary Algorithms: A Survey. *ACM Computing Surveys (CSUR)*, 45(3):35, 2013.

[61] P. Darwen and X. Yao. On Evolving Robust Strategies for Iterated Prisoner's Dilemma. *Progress in Evolutionary Computation*, pages 276–292, 1995.

[62] P Darwen and X Yao. Does Extra Genetic Diversity Maintain Escalation in a Co-evolutionary Arms Race. *International Journal of Knowledge-Based Intelligent Engineering Systems*, 4(3): 191–200, 2000.

[63] P.J. Darwen and X. Yao. Why More Choices Cause Less Cooperation in Iterated Prisoner's Dilemma. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 2, pages 987–994. IEEE, 2001.

[64] P.J. Darwen and X. Yao. Co-evolution in Iterated Prisoner's Dilemma with Intermediate Levels of Cooperation: Application to Missile Defense. *International Journal of Computational Intelligence and Applications*, 2:83–108, 2002.

[65] C. Darwin. On the Origin of Species by Means of Natural Selection. 1859. *Leipzig: Verlag Philipp Reclam*, 1984.

[66] Rajarshi Das, James P Crutchfield, Melanie Mitchell, and James M Hanson. Evolving Globally Synchronized Cellular Automata. 1995.

[67] E. de Jong. The Incremental Pareto-coevolution Archive. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 525–536. Springer, 2004.

[68] E. De Jong. The Maxsolve Algorithm for Coevolution. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 483–489. ACM, 2005.

[69] E.D. De Jong and A. Bucci. Deca: Dimension Extracting Coevolutionary Algorithm. In *Proceedings of the 8^{th} annual conference on Genetic and evolutionary computation*, pages 313–320. ACM, 2006.

[70] E.D. De Jong and J.B. Pollack. Ideal Evaluation from Coevolution. *Evolutionary Computation*, 12 (2):159–192, 2004.

[71] Edwin D De Jong. A Monotonic Archive for Pareto-coevolution. *Evolutionary computation*, 15(1): 61–93, 2007.

[72] Edwin D De Jong and Tim Oates. A Coevolutionary Approach to Representation Development. In *Proc. of the ICML-2002 Workshop on Development of Representations*, 2002.

[73] Kalyanmoy Deb. *Multi-objective Optimization Using Evolutionary Algorithms*, volume 16. John Wiley & Sons, 2001.

[74] Kalyanmoy Deb and Himanshu Jain. An Evolutionary Many-objective Optimization Algorithm Using Reference-point-based Nondominated Sorting Approach, Part I: Solving Problems with Box Constraints. *IEEE Trans. Evolutionary Computation*, 18(4):577–601, 2014.

[75] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2): 182–197, 2002.

[76] Kalyanmoy Deb, Karthik Sindhya, and Jussi Hakanen. Multi-objective Optimization. In *Decision Sciences: Theory and Practice*, pages 145–184. CRC Press, 2016.

[77] Karthik Devarajan. Nonnegative Matrix Factorization: An Analytical and Interpretive Tool in Computational Biology. *PLoS computational biology*, 4(7):e1000029, 2008.

[78] Włodzisław Duch. What Is Computational Intelligence and Where Is It Going? In *Challenges for computational intelligence*, pages 1–13. Springer, 2007.

[79] Adam Dziuk and Risto Miikkulainen. Creating Intelligent Agents through Shaping of Coevolution. In Alice E. Smith, editor, *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1077–1083, New Orleans, LA, USA, 2011. IEEE Press.

[80] Agoston E Eiben, James E Smith, et al. *Introduction to Evolutionary Computing*. Springer, 2015.

[81] Achiya Elyasaf, Ami Hauptman, and Moshe Sipper. Evolutionary Design of FreeCell Solvers. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:270–281, December 2012. ISSN 1943-068X. doi: 10.1109/TCIAIG.2012.2210423. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6249736.

[82] A.P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley, 2007.

[83] Susan L. Epstein. Toward an Ideal Trainer. *Machine Learning*, 15(3):251–277, 1994.

[84] Cesar Estebanez, Yago Saez, Gustavo Recio, and Pedro Isasi. Automatic Design of Noncryptographic Hash Functions Using Genetic Programming. *Computational Intelligence*. ISSN 1467-8640. doi: 10.1002/coin.12033. Early View (Online Version of Record published before inclusion in an issue).

[85] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level Classification of Skin Cancer with Deep Neural Networks. *Nature*, 542(7639):115, 2017.

[86] S. Ficici and J. Pollack. Pareto Optimality in Coevolutionary Learning. *Advances in Artificial Life*, pages 316–325, 2001.

[87] Sevan G Ficici. Monotonic Solution Concepts in Coevolution. In *Proceedings of the 7^{th} annual conference on Genetic and evolutionary computation*, pages 499–506. ACM, 2005.

[88] Sevan G Ficici and Jordan B Pollack. A Game-theoretic Approach to the Simple Coevolutionary Algorithm. In *International Conference on Parallel Problem Solving from Nature*, pages 467–476. Springer, 2000.

[89] Sevan Gregory Ficici. *Solution Concepts in Coevolutionary Algorithms*. PhD thesis, Computer Science Department, Brandeis University, USA, May 2004. URL http://www.demo.cs.brandeis.edu/papers/long.html#ficici_thesis_04.

[90] S.G. Ficici. Multiobjective Optimization and Coevolution. *Multiobjective Problem Solving from Nature*, pages 31–52, 2008.

[91] D. Fogel and L. Fogel. An Introduction to Evolutionary Programming. In *Artificial Evolution*, pages 21–33. Springer, 1996.

[92] David B Fogel. The Evolution of Intelligent Decision Making in Gaming. *Cybernetics and Systems*, 22(2):223–236, 1991.

[93] David B Fogel. *Blondie24: Playing at the Edge of Ai*. Morgan Kaufmann, 2001.

[94] David B Fogel, Timothy J Hays, Sarah L Hahn, James Quon, and G Kendall. Further Evolution of a Self-learning Chess Program. In *CIG*. Citeseer, 2005.

[95] David B Fogel, Timothy J Hays, Sarah L Hahn, and James Quon. The Blondie25 Chess Program Competes against Fritz 8.0 and a Human Chess Master. In *Computational Intelligence and Games, 2006 IEEE Symposium on*, pages 230–235. IEEE, 2006.

[96] D.B. Fogel and Z. Michalewicz. *Handbook of Evolutionary Computation*. Taylor & Francis, 1997.

[97] D.B. Fogel, E.C. Wasson, and E.M. Boughton. Evolving Neural Networks for Detecting Breast Cancer. *Cancer letters*, 96(1):49–53, 1995.

[98] D.B. Fogel, E.C. Wasson III, E.M. Boughton, and V.W. Porto. Evolving Artificial Neural Networks for Screening Features from Mammograms. *Artificial Intelligence in Medicine*, 14(3):317–326, 1998.

[99] L.J. Fogel. *On the Organization of Intellect*. PhD thesis, 1964.

[100] Carlos M Fonseca, Peter J Fleming, et al. Genetic Algorithms for Multiobjective Optimization: Formulationdiscussion and Generalization. In *Icga*, volume 93, pages 416–423. Citeseer, 1993.

[101] Stephanie Forrest, ThanhVu Nguyen, Westley Weimer, and Claire Le Goues. A Genetic Programming Approach to Automated Software Repair. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 947–954. ACM, 2009.

[102] Michael P Fourman. Compaction of Symbolic Layout Using Genetic Algorithms. In *Genetic Algorithms and Their Applications: Proc. 1st Int. Conf. Genetic Algorithms, Princeton, Lawrence Erlbaum, NJ, 1985*, 1985.

[103] M Frean. The evolution of degrees of cooperation. *Journal of theoretical biology*, 182(4):549–59, October 1996. ISSN 0022-5193. doi: 10.1006/jtbi.1996.0194. URL http://www.ncbi.nlm.nih.gov/pubmed/8944899.

[104] R. M. Friedberg. A Learning Machine: I. *IBM Journal of Research and Development*, 2(1):2–13, January 1958. ISSN 0018-8646. URL http://www.research.ibm.com/journal/rd/021/ibmrd0201B.pdf.

[105] John Fulcher. Computational Intelligence: An Introduction. In *Computational intelligence: a compendium*, pages 3–78. Springer, 2008.

[106] Pablo Garcia-Sanchez, Alberto Tonda, Antonio Mora, Giovanni Squillero, and J. J. Merelo. Towards Automatic Starcraft Strategy Generation Using Genetic Programming. In Shi-Jim Yen, Tristan Cazenave, and Philip Hingston, editors, *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG-2015)*, pages 284–291, Tainan, Taiwan, August 2015. IEEE. doi: 10.1109/CIG.2015.7317940. URL http://www.human-competitive.org/sites/default/files/garcia-sanchez-merelo-mora-squillero-tonda-text.txt.

[107] Pablo García-Sánchez, Alberto Tonda, Giovanni Squillero, Antonio Mora, and Juan J Merelo. Evolutionary Deckbuilding in Hearthstone. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pages 1–8. IEEE, 2016.

[108] Renaud Gaujoux and Cathal Seoighe. A Flexible R Package for Nonnegative Matrix Factorization. *BMC Bioinformatics*, 11(1):367, 2010. ISSN 1471-2105. doi: 10.1186/1471-2105-11-367. URL http://www.biomedcentral.com/1471-2105/11/367.

[109] Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. Large-scale Matrix Factorization with Distributed Stochastic Gradient Descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM, 2011.

[110] Robert Gibbons. *A Primer in Game Theory*. Harvester Wheatsheaf, 1992.

[111] Nicolas Gillis. Sparse and Unique Nonnegative Matrix Factorization through Data Preprocessing. *Journal of Machine Learning Research*, 13(Nov):3349–3386, 2012.

[112] Nicolas Gillis. The Why and How of Nonnegative Matrix Factorization. *Regularization, Optimization, Kernels, and Support Vector Machines*, 12(257), 2014.

[113] Nicolas Gillis and François Glineur. Accelerated Multiplicative Updates and Hierarchical Als Algorithms for Nonnegative Matrix Factorization. *Neural computation*, 24(4):1085–1105, 2012.

[114] Herbert Gintis. *Game Theory Evolving: A Problem-centered Introduction to Modeling Strategic Behavior*. Princeton university press, 2000.

[115] David E Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. *Addion wesley*, 1989:102, 1989.

[116] Faustino Gomez and Risto Miikkulainen. Incremental Evolution of Complex General Behavior. *Adaptive Behavior*, 5(3-4):317–342, 1997.

[117] Steven Gustafson and Leonardo Vanneschi. Crossover-based Tree Distance in Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 12(4):506–524, August 2008. ISSN 1089-778X. doi: 10.1109/TEVC.2008.915993.

[118] Prabhat Hajela and C-Y Lin. Genetic Search Strategies in Multicriterion Optimal Design. *Structural optimization*, 4(2):99–107, 1992.

[119] Jian Han, Lixing Han, Michael Neumann, and Upendra Prasad. On the Rate of Convergence of the Image Space Reconstruction Algorithm. *Operators and matrices*, 3(1):41–58, 2009.

[120] Nikolaus Hansen. The Cma Evolution Strategy: A Comparing Review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006.

[121] Paul G Harrald and David B Fogel. Evolving Continuous Behaviors in the Iterated Prisoner's Dilemma. *Biosystems*, 37(1):135–145, 1996.

[122] Inman Harvey, Philip Husbands, and David Cliff. *Seeing the Light: Artificial Evolution, Real Vision*. School of Cognitive and Computing Sciences, University of Sussex Falmer, 1994.

[123] Ami Hauptman and Moshe Sipper. Evolution of an Efficient Search Algorithm for the Mate-in-N Problem in Chess. In Marc Ebner, Michael O'Neill, Anikó Ekárt, Leonardo Vanneschi, and Anna Isabel Esparcia-Alcázar, editors, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 78–89, Valencia, Spain, 11-13 April 2007. Springer. ISBN 3-540-71602-5. doi: 10.1007/978-3-540-71605-1_8.

[124] Ami Hauptman, Achiya Elyasaf, Moshe Sipper, and Assaf Karmon. Gp-rush: Using Genetic Programming to Evolve Solvers for the Rush Hour Puzzle. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 955–962. ACM, 2009.

[125] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[126] Thomas Helmuth and Lee Spector. General Program Synthesis Benchmark Suite. In Sara Silva, Anna I Esparcia-Alcazar, Manuel Lopez-Ibanez, Sanaz Mostaghim, Jon Timmis, Christine Zarges, Luis Correia, Terence Soule, Mario Giacobini, Ryan Urbanowicz, Youhei Akimoto, Tobias Glasmachers, Francisco Fernandez de Vega, Amy Hoover, Pedro Larranaga, Marta Soto, Carlos Cotta, Francisco B. Pereira, Julia Handl, Jan Koutnik, Antonio Gaspar-Cunha, Heike Trautmann, Jean-Baptiste Mouret, Sebastian Risi, Ernesto Costa, Oliver Schuetze, Krzysztof Krawiec, Alberto Moraglio, Julian F. Miller, Pawel Widera, Stefano Cagnoni, JJ Merelo, Emma Hart, Leonardo Trujillo, Marouane Kessentini, Gabriela Ochoa, Francisco Chicano, and Carola Doerr, editors, *GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1039–1046, Madrid, Spain, 11-15 July 2015. ACM. doi: 10.1145/2739480.2754769.

[127] Thomas Helmuth, Lee Spector, and James Matheson. Solving Uncompromising Problems with Lexicase Selection. *IEEE Transactions on Evolutionary Computation*, 19(5):630–643, October 2015. ISSN 1089-778X. doi: 10.1109/TEVC.2014.2362729. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6920034.

[128] Thomas Helmuth, Nicholas Freitag McPhee, and Lee Spector. The Impact of Hyperselection on Lexicase Selection. In Tobias Friedrich, editor, *GECCO '16: Proceedings of the 2016 Annual Conference on Genetic and Evolutionary Computation*, pages 717–724, Denver, USA, 20-24 July 2016. ACM. doi: 10.1145/2908812.2908851. Nominated for best paper.

[129] Thomas M. Helmuth. *General Program Synthesis from Examples Using Genetic Programming with Parent Selection Based on Random Lexicographic Orderings of Test Cases*. PhD thesis, College of Information and Computer Sciences, University of Massachusetts Amherst, USA, September 2015. URL https://web.cs.umass.edu/publication/details.php?id=2398.

[130] Michael Hemesath. Survey Article: Cooperate or Defect? Russian and American Students in a Prisoner's Dilemma. *Comparative Economic Studies*, 36(1):83–93, 1994.

[131] Torsten Hildebrandt and Juergen Branke. On Using Surrogates with Genetic Programming. *Evolutionary Computation*, 23(3):343–367, Fall 2015. ISSN 1063-6560. doi: 10.1162/EVCO_a_00133.

[132] W Daniel Hillis. Co-evolving Parasites Improve Simulated Evolution As an Optimization Procedure. *Physica D: Nonlinear Phenomena*, 42(1-3):228–234, 1990.

[133] Geoffrey E Hinton and Richard S Zemel. Autoencoders, Minimum Description Length and Helmholtz Free Energy. In *Advances in neural information processing systems*, pages 3–10, 1994.

[134] John H. Holland. Adaptation in Natural and Artificial Systems. 1975.

[135] Myles Hollander, Douglas A Wolfe, and Eric Chicken. *Nonparametric Statistical Methods*, volume 751. John Wiley & Sons, 2013.

[136] Ting Hu, Joshua Payne, Jason Moore, and Wolfgang Banzhaf. Robustness, Evolvability, and Accessibility in Linear Genetic Programming. In Sara Silva, James A. Foster, Miguel Nicolau, Mario Giacobini, and Penousal Machado, editors, *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, volume 6621 of *LNCS*, pages 13–24, Turin, Italy, 27-29 April 2011. Springer Verlag. doi: 10.1007/978-3-642-20407-4_2.

[137] Kejun Huang, Nicholas D Sidiropoulos, and Ananthram Swami. Non-negative Matrix Factorization Revisited: Uniqueness and Algorithm for Symmetric Decomposition. *IEEE Transactions on Signal Processing*, 62(1):211–224, 2014.

[138] Phil Husbands and Frank Mill. Simulated Co-evolution As the Mechanism for Emergent Planning and Scheduling. In *ICGA*, pages 264–270, 1991.

[139] Hisao Ishibuchi and Tadahiko Murata. Multi-objective Genetic Local Search Algorithm. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 119–124. IEEE, 1996.

[140] Hisao Ishibuchi, Ryo Imada, Yu Setoguchi, and Yusuke Nojima. Performance Comparison of Nsga-ii and Nsga-iii on Various Many-objective Test Problems. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*, pages 3045–3052. IEEE, 2016.

[141] Wojciech Jaskowski. *Algorithms for Test-based Problems*. PhD thesis, Institute of Computing Science, Poznan University of Technology, Poznan, Poland, May 2011. URL http://www.cs.put.poznan.pl/wjaskowski/pub/papers/jaskowski11algorithms.pdf.

[142] Wojciech Jaśkowski. Systematic N-tuple Networks for Othello Position Evaluation. *ICGA Journal*, 37(2):85–96, 2014.

[143] Wojciech Jaśkowski and Krzysztof Krawiec. Coordinate System Archive for Coevolution. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–10. IEEE, 2010.

[144] Wojciech Jaśkowski and Krzysztof Krawiec. Formal Analysis, Hardness, and Algorithms for Extracting Internal Structure of Test-based Problems. *Evolutionary computation*, 19(4):639–671, 2011.

[145] Wojciech Jaskowski and Krzysztof Krawiec. How Many Dimensions in Co-optimization. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 829–830. ACM, 2011.

[146] Wojciech Jaśkowski and Marcin Szubert. Coevolutionary CMA-ES for Knowledge-free Learning of Game Position Evaluation. *IEEE Transactions on Computational Intelligence and AI in Games*, 8 (4):389–401, 2016. ISSN 1943-0698. doi: 10.1109/TCIAIG.2015.2464711. URL http://www.cs.put.poznan.pl/wjaskowski/pub/papers/Jaskowski2015CoCMAES.pdf.

[147] Wojciech Jaśkowski, Krzysztof Krawiec, and Bartosz Wieloch. Evolving Strategy for a Probabilistic Game of Imperfect Information Using Genetic Programming. *Genetic Programming and Evolvable Machines*, 9(4):281–294, December 2008. ISSN 1389-2576. doi: 10.1007/s10710-008-9062-1.

[148] Wojciech Jaskowski, Krzysztof Krawiec, and Bartosz Wieloch. Winning Ant Wars: Evolving a Human-competitive Game Strategy Using Fitnessless Selection. In Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, Anna Isabel Esparcia Alcazar, Ivanoe De Falco, Antonio Della Cioppa, and Ernesto Tarantino, editors, *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, volume 4971 of *Lecture Notes in Computer Science*, pages 13–24, Naples, 26-28 March 2008. Springer. doi: 10.1007/978-3-540-78671-9_2.

[149] Wojciech Jaśkowski, Paweł Liskowski, Marcin Szubert, and Krzysztof Krawiec. Improving Coevolution by Random Sampling. In Christian Blum, editor, *GECCO'13: Proceedings of the 15th annual conference on Genetic and Evolutionary Computation*, pages 1141–1148, Amsterdam, The Netherlands, July 2013. ACM. URL http://www.cs.put.poznan.pl/mszubert/pub/jaskowski2013gecco.pdf.

[150] Wojciech Jaśkowski, Marcin Szubert, and Paweł Liskowski. Multi-criteria Comparison of Coevolution and Temporal Difference Learning on Othello. In A. I. Esparcia-Alcazar and A. M. Mora, editors, *EvoApplications 2014*, volume 8602 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2014.

[151] Wojciech Jaśkowski, Paweł Liskowski, Marcin Szubert, and Krzysztof Krawiec. Performance Profile: A Multi-criteria Performance Evaluation Method for Test-based Problems. *International Journal of Applied Mathematics and Computer Science*, 26(1):215–229, 2016. doi: 10.1515/amcs-2016-0015.

[152] Mikkel T Jensen. Helper-objectives: Using Multi-objective Evolutionary Algorithms for Single-objective Optimisation. *Journal of Mathematical Modelling and Algorithms*, 3(4):323–347, 2004.

[153] Yaochu Jin. Surrogate-assisted Evolutionary Computation: Recent Advances and Future Challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

[154] Yaochu Jin and J. Branke. Evolutionary Optimization in Uncertain Environments-a Survey. *Evolutionary Computation, IEEE Transactions on*, 9(3):303–317, 2005. ISSN 1089-778X. doi: 10.1109/TEVC.2005.846356. Survey of noisy environments.

[155] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient Global Optimization of Expensive Black-box Functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[156] Hugues Juillé. *Methods for Statistical Inference: Extending the Evolutionary Computation Paradigm*. PhD thesis, Waltham, MA, USA, 1999.

[157] Hugues Juille and Jordan B. Pollack. Coevolving the Ideal Trainer: Application to the Discovery of Cellular Automata Rules. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 519–527, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann. URL http://www.demo.cs.brandeis.edu/papers/gp98.pdf.

[158] Uday Kamath, Jack Compton, Rezarta Islamaj-Doğan, Kenneth A De Jong, and Amarda Shehu. An Evolutionary Algorithm Approach for Feature Generation from Sequence Data and Its Application to Dna Splice Site Prediction. *IEEE/ACM transactions on computational biology and bioinformatics*, 9(5):1387–1398, 2012.

[159] Gopal K Kanji. *100 Statistical Tests*. Sage, 2006.

[160] Frédéric Kaplan and Verena V Hafner. Information-theoretic Framework for Unsupervised Activity Classification. *Advanced Robotics*, 20(10):1087–1103, 2006.

[161] Jan Karasek, Radim Burget, and Ondrej Morsky. Towards an Automatic Design of Non-cryptographic Hash Function. In *34^{th} International Conference on Telecommunications and Signal Processing (TSP 2011)*, pages 19–23, Budapest, 18-20 August 2011. doi: 10.1109/TSP.2011.6043785.

[162] Vineet Khare, Xin Yao, and Kalyanmoy Deb. Performance Scaling of Multi-objective Evolutionary Algorithms. In *Evolutionary Multi-Criterion Optimization*, pages 376–390. Springer, 2003.

[163] Lemont B Kier, Paul G Seybold, and Chao-Kun Cheng. *Modeling Chemical Systems Using Cellular Automata*, volume 1. Springer Science & Business Media, 2005.

[164] Jae Yun Kim, Yeo Keun Kim, and Yeongho Kim. Tournament Competition and Its Merits for Coevolutionary Algorithms. *Journal of Heuristics*, 9(3):249–268, 2003.

[165] Kenneth E Kinnear. Fitness Landscapes and Difficulty in Genetic Programming. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 142–147. IEEE, 1994.

[166] Joshua Knowles and David Corne. Quantifying the Effects of Objective Space Dimension in Evolutionary Multiobjective Optimization. In *Evolutionary Multi-Criterion Optimization*, pages 757–771. Springer, 2007.

[167] Joshua D Knowles and David W Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary computation*, 8(2):149–172, 2000.

[168] Joshua D Knowles, Richard A Watson, and David W Corne. Reducing Local Optima in Single-objective Problems by Multi-objectivization. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 269–283. Springer, 2001.

[169] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8), 2009.

[170] Derrick G Kourie and Bruce W Watson. *The Correctness-by-construction Approach to Programming*. Springer Science & Business Media, 2012.

[171] J. Koza. Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems. Technical Report STAN-CS-90-1314, Dept. of Computer Science, Stanford University, June 1990. URL http://www.genetic-programming.com/jkpdf/tr1314.pdf.

[172] J. R. Koza. Hierarchical Genetic Algorithms Operating on Populations of Computer Programs. In N. S. Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI-89*, volume 1, pages 768–774, Detroit, MI, USA, 20-25 August 1989. Morgan Kaufmann. URL http://www.genetic-programming.com/jkpdf/ijcai1989.pdf.

[173] John R. Koza. A Hierarchical Approach to Learning the Boolean Multiplexer Function. In Gregory J. E. Rawlins, editor, *Foundations of genetic algorithms*, pages 171–192. Morgan Kaufmann, Indiana University, 15-18 July 1990 1991. URL http://www.genetic-programming.com/jkpdf/foga1990.pdf.

[174] John R. Koza. Concept Formation and Decision Tree Induction Using the Genetic Programming Paradigm. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, volume 496 of *Lecture Notes in Computer Science*, pages 124–128, Dortmund, Germany, 1-3 October 1991. Springer-Verlag. URL http://www.genetic-programming.com/jkpdf/ppsn1990.pdf.

[175] John R. Koza. Genetic Evolution and Co-evolution of Game Strategies. In *The International Conference on Game Theory and Its Applications*, Stony Brook, New York, July 1992. URL http://www.genetic-programming.com/jkpdf/icgt1992.pdf.

[176] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-11170-5. URL http://mitpress.mit.edu/books/genetic-programming.

[177] John R. Koza and David Andre. Automatic Discovery of Protein Motifs Using Genetic Programming. In Xin Yao, editor, *Evolutionary Computation: Theory and Applications*, chapter 5, pages 171–197. World Scientific, Singapore, 1999. ISBN 981-02-2306-4. URL http://www.genetic-programming.com/jkpdf/ecta1999.pdf.

[178] John R. Koza and James P. Rice. Genetic Generation of Both the Weights and Architecture for a Neural Network. In *International Joint Conference on Neural Networks, IJCNN-91*, volume II, pages 397–404, Washington State Convention and Trade Center, Seattle, WA, USA, 8-12 July 1991. IEEE Computer Society Press. ISBN 0-7803-0164-1. doi: 10.1109/IJCNN.1991.155366. URL http://www.genetic-programming.com/jkpdf/ijcnn1991.pdf.

[179] John R. Koza, Forrest H Bennett III, David Andre, and Martin A. Keane. Four Problems for Which a Computer Program Evolved by Genetic Programming Is Competitive with Human Performance. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, volume 1, pages 1–10. IEEE Press, 1996. URL http://www.genetic-programming.com/jkpdf/icec1996.pdf.

[180] John R. Koza, Forrest H Bennett III, David Andre, and Martin A Keane. Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming. In John S. Gero and Fay Sudweeks, editors, *Artificial Intelligence in Design '96*, pages 151–170, Dordrecht, 1996. Kluwer Academic. URL http://www.genetic-programming.com/jkpdf/aid1996.pdf.

[181] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-competitive Machine Intelligence*. Kluwer Academic Publishers, 2003. ISBN 1-4020-7446-8. URL http://www.genetic-programming.org/gpbook4toc.html.

[182] Krzysztof Krawiec. Genetic Programming-based Construction of Features for Machine Learning and Knowledge Discovery Tasks. *Genetic Programming and Evolvable Machines*, 3(4):329–343, December 2002. ISSN 1389-2576. doi: 10.1023/A:1020984725014.

[183] Krzysztof Krawiec. *Behavioral Program Synthesis with Genetic Programming*, volume 618 of *Studies in Computational Intelligence*. Springer International Publishing, 2016. ISBN 978-3-319-27563-5. doi: 10.1007/978-3-319-27565-9. URL http://www.springer.com/gp/book/9783319275635.

[184] Krzysztof Krawiec and Bir Bhanu. Visual Learning by Coevolutionary Feature Synthesis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(3):409–425, 2005.

[185] Krzysztof Krawiec and Paweł Lichocki. Using Co-solvability to Model and Exploit Synergetic Effects in Evolution. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Guenter Rudolph, editors, *PPSN 2010 11th International Conference on Parallel Problem Solving From Nature*, volume 6239 of *Lecture Notes in Computer Science*, pages 492–501, Krakow, Poland, 11-15 September 2010. Springer. doi: 10.1007/978-3-642-15871-1_50.

[186] Krzysztof Krawiec and Paweł Liskowski. Automatic Derivation of Search Objectives for Test-based Genetic Programming. In Penousal Machado, Malcolm I. Heywood, James McDermott, Mauro Castelli, Pablo Garcia-Sanchez, Paolo Burelli, Sebastian Risi, and Kevin Sim, editors, *18th European Conference on Genetic Programming*, volume 9025 of *LNCS*, pages 53–65, Copenhagen, 8-10 April 2015. Springer. doi: 10.1007/978-3-319-16501-1_5.

[187] Krzysztof Krawiec and Una-May O'Reilly. Behavioral Programming: A Broader and More Detailed Take on Semantic GP. In Christian Igel, Dirk V. Arnold, Christian Gagne, Elena Popovici, Anne Auger, Jaume Bacardit, Dimo Brockhoff, Stefano Cagnoni, Kalyanmoy Deb, Benjamin Doerr, James Foster, Tobias Glasmachers, Emma Hart, Malcolm I. Heywood, Hitoshi Iba, Christian Jacob, Thomas Jansen, Yaochu Jin, Marouane Kessentini, Joshua D. Knowles, William B. Langdon, Pedro Larranaga, Sean Luke, Gabriel Luque, John A. W. McCall, Marco A. Montes de Oca, Alison Motsinger-Reif, Yew Soon Ong, Michael Palmer, Konstantinos E. Parsopoulos, Guenther Raidl, Sebastian Risi, Guenther Ruhe, Tom Schaul, Thomas Schmickl, Bernhard Sendhoff, Kenneth O. Stanley, Thomas Stuetzle, Dirk Thierens, Julian Togelius, Carsten Witt, and Christine Zarges, editors, *GECCO '14: Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 935–942, Vancouver, BC, Canada, 12-16 July 2014. ACM. doi: 10.1145/2576768.2598288. Best paper.

[188] Krzysztof Krawiec and Una-May O'Reilly. Behavioral Search Drivers for Genetic Programing. In Miguel Nicolau, Krzysztof Krawiec, Malcolm I. Heywood, Mauro Castelli, Pablo Garcia-Sanchez, Juan J. Merelo, Victor M. Rivas Santos, and Kevin Sim, editors, *17th European Conference on Genetic Programming*, volume 8599 of *LNCS*, pages 210–221, Granada, Spain, 23-25 April 2014. Springer. doi: 10.1007/978-3-662-44303-3_18.

[189] Krzysztof Krawiec and Mikolaj Pawlak. Genetic Programming with Alternative Search Drivers for Detection of Retinal Blood Vessels. In Antonio M. Mora and Giovanni Squillero, editors, *18th European Conference on the Applications of Evolutionary Computation*, volume 9028 of *LNCS*, pages 554–566, Copenhagen, 8-10 April 2015. Springer. doi: 10.1007/978-3-319-16549-3_45.

[190] Krzysztof Krawiec and Tomasz Pawlak. Locally Geometric Semantic Crossover. In Terry Soule, Anne Auger, Jason Moore, David Pelta, Christine Solnon, Mike Preuss, Alan Dorin, Yew-Soon Ong, Christian Blum, Dario Landa Silva, Frank Neumann, Tina Yu, Aniko Ekart, Will Browne, Tim Kovacs, Man-Leung Wong, Clara Pizzuti, Jon Rowe, Tobias Friedrich, Giovanni Squillero, Nicolas Bredeche, Stephen L. Smith, Alison Motsinger-Reif, Jose Lozano, Martin Pelikan, Silja Meyer-Nienberg, Christian Igel, Greg Hornby, Rene Doursat, Steve Gustafson, Gustavo Olague, Shin Yoo, John Clark, Gabriela Ochoa, Gisele Pappa, Fernando Lobo, Daniel Tauritz, Jurgen Branke, and Kalyanmoy Deb, editors, *GECCO Companion '12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pages 1487–1488, Philadelphia, Pennsylvania, USA, 7-11 July 2012. ACM. doi: 10.1145/2330784.2331005.

[191] Krzysztof Krawiec and Armando Solar-Lezama. Improving Genetic Programming with Behavioral Consistency Measure. In Thomas Bartz-Beielstein, Juergen Branke, Bogdan Filipic, and Jim Smith, editors, *13th International Conference on Parallel Problem Solving from Nature*, volume 8672 of *Lecture Notes in Computer Science*, pages 434–443, Ljubljana, Slovenia, 13-17 September 2014. Springer. doi: 10.1007/978-3-319-10762-2_43.

[192] Krzysztof Krawiec and Jerry Swan. Pattern-guided Genetic Programming. In Christian Blum, Enrique Alba, Anne Auger, Jaume Bacardit, Josh Bongard, Juergen Branke, Nicolas Bredeche, Dimo Brockhoff, Francisco Chicano, Alan Dorin, Rene Doursat, Aniko Ekart, Tobias Friedrich, Mario Giacobini, Mark Harman, Hitoshi Iba, Christian Igel, Thomas Jansen, Tim Kovacs, Taras Kowaliw, Manuel Lopez-Ibanez, Jose A. Lozano, Gabriel Luque, John McCall, Alberto Moraglio, Alison Motsinger-Reif, Frank Neumann, Gabriela Ochoa, Gustavo Olague, Yew-Soon Ong, Michael E. Palmer, Gisele Lobo Pappa, Konstantinos E. Parsopoulos, Thomas Schmickl, Stephen L. Smith, Christine Solnon, Thomas Stuetzle, El-Ghazali Talbi, Daniel Tauritz, and Leonardo Vanneschi, editors, *GECCO '13: Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 949–956, Amsterdam, The Netherlands, 6-10 July 2013. ACM. doi: 10.1145/2463372.2463496.

[193] Krzysztof Krawiec, Wojciech Jaśkowski, and Marcin Szubert. Evolving Small-board Go Players Using Coevolutionary Temporal Difference Learning with Archive. *International Journal of Applied Mathematics and Computer Science*, 21(4):717–731, 2011.

[194] Krzysztof Krawiec, Jerry Swan, and Una-May O'Reilly. Behavioral Program Synthesis: Insights and Prospects. In *Genetic Programming Theory and Practice XIII*, pages 169–183. Springer, 2016.

[195] Frank Kursawe. A Variant of Evolution Strategies for Vector Optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 193–197. Springer, 1990.

[196] Ray Kurzweil. *How to Create a Mind: The Secret of Human Thought Revealed*. Penguin, 2013.

[197] William La Cava, Lee Spector, and Kourosh Danai. Epsilon-lexicase Selection for Regression. In Tobias Friedrich, editor, *GECCO '16: Proceedings of the 2016 Annual Conference on Genetic and Evolutionary Computation*, pages 741–748, Denver, USA, 20-24 July 2016. ACM. doi: 10.1145/2908812.2908898.

[198] W. B. Langdon. Quadratic Bloat in Genetic Programming. In Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 451–458, Las Vegas, Nevada, USA, 10-12 July 2000. Morgan Kaufmann. ISBN 1-55860-708-0. URL http://www.cs.bham.ac.uk/~wbl/biblio/gecco2000/GA069.pdf.

[199] W. B. Langdon and S. M. Gustafson. Genetic Programming and Evolvable Machines: Ten Years of Reviews. *Genetic Programming and Evolvable Machines*, 11(3/4):321–338, September 2010. ISSN 1389-2576. doi: 10.1007/s10710-010-9111-4. URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/gppubs10.pdf. Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines.

[200] W. B. Langdon and J. P. Nordin. Seeding GP Populations. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 304–315, Edinburgh, 15-16 April 2000. Springer-Verlag. ISBN 3-540-67339-3. doi: 10.1007/978-3-540-46239-2_23. URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/WBL_eurogp2000_seed.pdf.

[201] William B. Langdon and Mark Harman. Genetically Improved CUDA C++ Software. In Miguel Nicolau, Krzysztof Krawiec, Malcolm I. Heywood, Mauro Castelli, Pablo Garcia-Sanchez, Juan J.

Merelo, Victor M. Rivas Santos, and Kevin Sim, editors, *17th European Conference on Genetic Programming*, volume 8599 of *LNCS*, pages 87–99, Granada, Spain, 23-25 April 2014. Springer. doi: 10.1007/978-3-662-44303-3_8. URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/langdon_2014_EuroGP.pdf.

[202] William B. Langdon, Marc Modat, Justyna Petke, and Mark Harman. Improving 3D Medical Image Registration CUDA Software with Genetic Programming. In Christian Igel, Dirk V. Arnold, Christian Gagne, Elena Popovici, Anne Auger, Jaume Bacardit, Dimo Brockhoff, Stefano Cagnoni, Kalyanmoy Deb, Benjamin Doerr, James Foster, Tobias Glasmachers, Emma Hart, Malcolm I. Heywood, Hitoshi Iba, Christian Jacob, Thomas Jansen, Yaochu Jin, Marouane Kessentini, Joshua D. Knowles, William B. Langdon, Pedro Larranaga, Sean Luke, Gabriel Luque, John A. W. McCall, Marco A. Montes de Oca, Alison Motsinger-Reif, Yew Soon Ong, Michael Palmer, Konstantinos E. Parsopoulos, Guenther Raidl, Sebastian Risi, Guenther Ruhe, Tom Schaul, Thomas Schmickl, Bernhard Sendhoff, Kenneth O. Stanley, Thomas Stuetzle, Dirk Thierens, Julian Togelius, Carsten Witt, and Christine Zarges, editors, *GECCO '14: Proceeding of the sixteenth annual conference on genetic and evolutionary computation conference*, pages 951–958, Vancouver, BC, Canada, 12-15 July 2014. ACM. doi: 10.1145/2576768.2598244. URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/Langdon_2014_GECCO.pdf.

[203] Christian W. G. Lasarczyk, Peter Dittrich, and Wolfgang Banzhaf. Dynamic Subset Selection Based on a Fitness Case Topology. *Evolutionary Computation*, 12(2):223–242, Summer 2004. doi: 10.1162/106365604773955157. URL http://ls11-www.cs.uni-dortmund.de/people/lasar/publication/LasarDittBanz_TBS_2004/LasarDittBanz_TBS_2004.pdf.

[204] Hans Laurberg, Mads Græsbøll Christensen, Mark D Plumbley, Lars Kai Hansen, and Søren Holdt Jensen. Theorems on Positive Data: On the Uniqueness of Nmf. *Computational intelligence and neuroscience*, 2008, 2008.

[205] Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, and Westley Weimer. A Systematic Study of Automated Program Repair: Fixing 55 Out of 105 Bugs for $8 Each. In Martin Glinz, editor, *34th International Conference on Software Engineering (ICSE 2012)*, pages 3–13, Zurich, June 2012. doi: 10.1109/ICSE.2012.6227211. URL http://dijkstra.cs.virginia.edu/genprog/papers/weimer-icse2012-genprog-preprint.pdf.

[206] Daniel D Lee and H Sebastian Seung. Learning the Parts of Objects by Non-negative Matrix Factorization. *Nature*, 401(6755):788–791, 1999.

[207] Daniel D Lee and H Sebastian Seung. Algorithms for Non-negative Matrix Factorization. In *Advances in neural information processing systems*, pages 556–562, 2001.

[208] Joel Lehman and Kenneth O Stanley. Abandoning Objectives: Evolution through the Search for Novelty Alone. *Evolutionary computation*, 19(2):189–223, 2011.

[209] Douglas B Lenat. The Role of Heuristics in Learning by Discovery: Three Case Studies. In *Machine learning*, pages 243–306. Springer, 1983.

[210] Paweł Liskowski and Wojciech Jaśkowski. Accelerating Coevolution with Adaptive Matrix Factorization, (nominated to Best-paper Award). In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 457–464, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4920-8. doi: 10.1145/3071178.3071320.

[211] Paweł Liskowski and Krzysztof Krawiec. Discovery of Implicit Objectives by Compression of Interaction Matrix in Test-based Problems. In *Parallel Problem Solving from Nature–PPSN XIII*, pages 611–620. Springer International Publishing, 2014.

[212] Paweł Liskowski and Krzysztof Krawiec. Surrogate Fitness Via Factorization of Interaction Matrix (Best-paper Award winner). In Malcolm I. Heywood, James McDermott, Mauro Castelli, and Ernesto Costa, editors, *EuroGP 2016: Proceedings of the 19th European Conference on Genetic Programming*, volume 9594 of *LNCS*, pages 65–79, Porto, Portugal, 30 March–1 April 2016. Springer Verlag.

[213] Paweł Liskowski and Krzysztof Krawiec. Non-negative Matrix Factorization for Unsupervised Derivation of Search Objectives in Genetic Programming. In Tobias Friedrich, editor, *GECCO '16: Proceedings of the 2016 Annual Conference on Genetic and Evolutionary Computation*, pages 749–756, Denver, USA, 20-24 July 2016. ACM. doi: 10.1145/2908812.2908888.

[214] Paweł Liskowski and Krzysztof Krawiec. Online Discovery of Search Objectives for Test-based Problems. *Evolutionary Computation*, 25(3):375–406, 2016.

[215] Paweł Liskowski and Krzysztof Krawiec. Segmenting Retinal Blood Vessels with Deep Neural Networks. *IEEE transactions on medical imaging*, 35(11):2369–2380, 2016.

[216] Paweł Liskowski and Krzysztof Krawiec. Discovery of Search Objectives in Continuous Domains. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 969–976, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4920-8. doi: 10.1145/3071178.3071344.

[217] Paweł Liskowski and Krzysztof Krawiec. Adaptive Test Selection for Factorization-based Surrogate Fitness in Genetic Programming. *Foundations of Computing and Decision Sciences*, 42(4):339–358, 2017.

[218] Paweł Liskowski, Krzysztof Krawiec, Thomas Helmuth, and Lee Spector. Comparison of Semantic-aware Selection Methods in Genetic Programming. In Colin Johnson, Krzysztof Krawiec, Alberto Moraglio, and Michael O'Neill, editors, *GECCO 2015 Semantic Methods in Genetic Programming (SMGP'15) Workshop*, pages 1301–1307, Madrid, Spain, 11-15 July 2015. ACM. doi: 10.1145/2739482.2768505.

[219] Paweł Liskowski, Wojciech Jaśkowski, and Krzysztof Krawiec. Learning to Play Othello with Deep Neural Networks. *IEEE Transactions on Games*, 2018.

[220] Darrell F Lochtefeld and Frank W Ciarallo. Helper-objective Optimization Strategies for the Job-shop Scheduling Problem. *Applied Soft Computing*, 11(6):4161–4174, 2011.

[221] Jason Lohn, Gregory Hornby, and Derek Linden. An Evolved Antenna for Deployment on Nasa's Space Technology 5 Mission. In Una-May O'Reilly, Tina Yu, Rick L. Riolo, and Bill Worzel, editors, *Genetic Programming Theory and Practice II*, chapter 18, pages 301–315. Springer, Ann Arbor, 13-15 May 2004. ISBN 0-387-23253-2. doi: 10.1007/0-387-23254-0_18.

[222] Antonio López Jaimes, Carlos A Coello Coello, and Debrup Chakraborty. Objective Reduction Using a Feature Selection Technique. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 673–680. ACM, 2008.

[223] Sushil J Louis and Gregory JE Rawlins. Pareto Optimality, Ga-easiness and Deception. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 118–123. Morgan Kaufmann Publishers Inc., 1993.

[224] Ada Lovelace. Notes on L. Menabre's Sketch of the Analytical Engine Invented by Charles Babbage. 1843.

[225] Alex Lubberts and Risto Miikkulainen. Co-evolving a Go-playing Neural Network. In *Proceedings of the GECCO-01 Workshop on Coevolution: Turning Adaptive Algorithms upon Themselves*, pages 14–19, 2001.

[226] Simon M Lucas and Thomas P Runarsson. Temporal Difference Learning Versus Co-evolution for Acquiring Othello Position Evaluation. In *Computational Intelligence and Games, 2006 IEEE Symposium on*, pages 52–59. IEEE, 2006.

[227] S.M. Lucas. Learning to Play Othello with N-tuple Systems. *Australian Journal of Intelligent Information Processing*, 4:1–20, 2008.

[228] Sean Luke. Genetic Programming Produced Competitive Soccer softbot teams for RoboCup97. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 214–222, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann. ISBN 1-55860-548-7. URL http://www.cs.gmu.edu/~sean/papers/robocupgp98.pdf.

[229] Sean Luke. Two Fast Tree-creation Algorithms for Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283, September 2000. URL http://ieeexplore.ieee.org/iel5/4235/18897/00873237.pdf.

[230] Sean Luke and Liviu Panait. A Survey and Comparison of Tree Generation algorithms. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 81–88, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann. ISBN 1-55860-774-9. URL http://www.cs.bham.ac.uk/~wbl/biblio/gecco2001/d01.pdf.

[231] Sean Luke and Liviu Panait. Lexicographic Parsimony Pressure. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 829–836, New York, 9-13 July 2002. Morgan Kaufmann Publishers. ISBN 1-55860-878-8. URL http://cs.gmu.edu/~sean/papers/lexicographic.pdf.

[232] Sean Luke and R. Paul Wiegand. Guaranteeing Coevolutionary Objective Measures. In Kenneth A. de Jong, Riccardo Poli, and Jonathan E. Rowe, editors, *Foundations of Genetic Algorithms VII*, pages 237–251, Torremolinos, Spain, 2002. Morgan Kaufman.

[233] Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, and James Hendler. Co-evolving Soccer Softbot Team Coordination with genetic programming. In *Proceedings of the First International Workshop on RoboCup, at the International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997. URL http://www.cs.gmu.edu/~sean/papers/robocupc.pdf.

[234] Jacek Mandziuk. *Knowledge-free and Learning-based Methods in Intelligent Game Playing*, volume 276. Springer, 2010.

[235] Zohar Manna and Richard Waldinger. A Deductive Approach to Program Synthesis. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 2(1):90–121, 1980.

[236] Edward P Manning. Coevolution in a Large Search Space Using Resource-limited Nash Memory. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 999–1006. ACM, 2010.

[237] Edward P Manning. Using Resource-limited Nash Memory to Improve an Othello Evaluation Function. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):40–53, 2010.

[238] James McDermott, David R. White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaskowski, Krzysztof Krawiec, Robin Harper, Kenneth De Jong, and

Una-May O'Reilly. Genetic Programming Needs Better Benchmarks. In Terry Soule, Anne Auger, Jason Moore, David Pelta, Christine Solnon, Mike Preuss, Alan Dorin, Yew-Soon Ong, Christian Blum, Dario Landa Silva, Frank Neumann, Tina Yu, Aniko Ekart, Will Browne, Tim Kovacs, Man-Leung Wong, Clara Pizzuti, Jon Rowe, Tobias Friedrich, Giovanni Squillero, Nicolas Bredeche, Stephen L. Smith, Alison Motsinger-Reif, Jose Lozano, Martin Pelikan, Silja Meyer-Nienberg, Christian Igel, Greg Hornby, Rene Doursat, Steve Gustafson, Gustavo Olague, Shin Yoo, John Clark, Gabriela Ochoa, Gisele Pappa, Fernando Lobo, Daniel Tauritz, Jurgen Branke, and Kalyanmoy Deb, editors, *GECCO '12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, pages 791–798, Philadelphia, Pennsylvania, USA, 7-11 July 2012. ACM. doi: 10.1145/2330163.2330273.

[239] Ben McKay, Mark J Willis, and Geoffrey W Barton. Using a Tree Structured Genetic Algorithm to Perform Symbolic Regression. In *Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995. GALESIA. First International Conference on (Conf. Publ. No. 414)*, pages 487–492. IET, 1995.

[240] R. I. (Bob) McKay. Committee Learning of Partial Functions in Fitness-shared Genetic Programming. In *Industrial Electronics Society, 2000. IECON 2000. 26th Annual Confjerence of the IEEE Third Asia-Pacific Conference on Simulated Evolution and Learning 2000*, volume 4, pages 2861–2866, Nagoya, Japan, October 2000. IEEE Press. ISBN 0-7803-6456-2. doi: 10.1109/IECON.2000.972452. URL http://sc.snu.ac.kr/PAPERS/committee.pdf.

[241] R I (Bob) McKay. Fitness Sharing in Genetic Programming. In Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 435–442, Las Vegas, Nevada, USA, 10-12 July 2000. Morgan Kaufmann. ISBN 1-55860-708-0. URL http://www.cs.bham.ac.uk/~wbl/biblio/gecco2000/GP256.pdf.

[242] Nicholas Freitag McPhee, Brian Ohs, and Tyler Hutchison. Semantic Building Blocks in Genetic Programming. In Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, Anna Isabel Esparcia Alcazar, Ivanoe De Falco, Antonio Della Cioppa, and Ernesto Tarantino, editors, *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, volume 4971 of *Lecture Notes in Computer Science*, pages 134–145, Naples, 26-28 March 2008. Springer. doi: 10.1007/978-3-540-78671-9_12.

[243] Zbigniew Michalewicz. *Genetic Algorithms+ Data Structures= Evolution Programs*. Springer, 1996.

[244] Ryszard S Michalski. A Theory and Methodology of Inductive Learning. *Artificial intelligence*, 20 (2):111–161, 1983.

[245] John H Miller. The Coevolution of Automata in the Repeated Prisoner's Dilemma. *Journal of Economic Behavior & Organization*, 29(1):87–112, 1996.

[246] Julian F. Miller, editor. *Cartesian Genetic Programming*. Natural Computing Series. Springer, 2011. doi: 10.1007/978-3-642-17310-3. URL http://www.springer.com/computer/theoretical+computer+science/book/978-3-642-17309-7.

[247] Marvin Minsky. Steps toward Artificial Intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[248] Tom M Mitchell. Machine Learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

[249] Tom M Mitchell, Paul E Utgoff, and Ranan Banerji. Learning by Experimentation: Acquiring and Refining Problem-solving Heuristics. In *Machine Learning*, pages 163–190. Springer, 1983.

[250] David J. Montana. Strongly Typed Genetic Programming. *Evolutionary Computation*, 3(2): 199–230, 1995. doi: 10.1162/evco.1995.3.2.199. URL http://vishnu.bbn.com/papers/stgp.pdf.

[251] Alberto Moraglio and Krzysztof Krawiec. Semantic Genetic Programming. In Anabela Simoes, editor, *GECCO 2015 Advanced Tutorials*, pages 603–627, Madrid, Spain, 11-15 July 2015. ACM. doi: 10.1145/2739482.2756587.

[252] Alberto Moraglio, Krzysztof Krawiec, and Colin G. Johnson. Geometric Semantic Genetic Programming. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature, PPSN XII (part 1)*, volume 7491 of *Lecture Notes in Computer Science*, pages 21–31, Taormina, Italy, September 2012. Springer. doi: 10.1007/978-3-642-32937-1_3.

[253] David E Moriarty and Risto Miikkulainen. Discovering Complex Othello Strategies through Evolutionary Neural Networks. *Connection Science*, 7(3-1):195–210, 1995.

[254] J-B Mouret and Stéphane Doncieux. Encouraging Behavioral Diversity in Evolutionary Robotics: An Empirical Study. *Evolutionary computation*, 20(1):91–133, 2012.

[255] Jean-Baptiste Mouret and Stéphane Doncieux. Incremental Evolution of Animats' Behaviors As a Multi-objective Optimization. In *International Conference on Simulation of Adaptive Behavior*, pages 210–219. Springer, 2008.

[256] Jason Noble. Finding Robust Texas Hold'em Poker Strategies Using Pareto Coevolution and Deterministic Crowding. 2002.

[257] Jason Noble and Richard A Watson. Pareto Coevolution: Using Performance against Coevolved Opponents in a Game As Dimensions for Pareto Selection. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 493–500. Morgan Kaufmann Publishers Inc., 2001.

[258] S. Nolfi and D. Floreano. Coevolving Predator and Prey Robots: Do "arms Races" Arise in Artificial Evolution? *Artificial Life*, 4(4):311–335, 1998.

[259] Martin A Nowak and Karl Sigmund. Evolutionary Dynamics of Biological Games. *science*, 303 (5659):793–799, 2004.

[260] Björn Olsson. Evaluation of a Simple Host-parasite Genetic Algorithm. In *Evolutionary Programming VII*, pages 53–62. Springer, 1998.

[261] Björn Olsson. Co-evolutionary Search in Asymmetric Spaces. *Information Sciences*, 133(3): 103–125, 2001.

[262] Una-May O'Reilly. Using a Distance Metric on Genetic Programs to understand genetic operators. In *IEEE International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation*, volume 5, pages 4092–4097, Orlando, Florida, USA, 12-15 October 1997. ISBN 0-7803-4053-1. URL http://ieeexplore.ieee.org/iel4/4942/13793/00637337.pdf.

[263] Michael Orlov and Moshe Sipper. Flight of the FINCH through the Java Wilderness. *IEEE Transactions on Evolutionary Computation*, 15(2):166–182, April 2011. ISSN 1089-778X. doi: 10.1109/TEVC.2010.2052622.

[264] Martin J Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT press, 1994.

[265] P-Y Oudeyer, Frédéric Kaplan, and Verena Vanessa Hafner. Intrinsic Motivation Systems for Autonomous Mental Development. *Evolutionary Computation, IEEE Transactions on*, 11(2): 265–286, 2007.

[266] Ovid and Charles Martin. *Metamorphoses*. W.W. Norton, 2004.

[267] Pentti Paatero and Unto Tapper. Positive Matrix Factorization: A Non-negative Factor Model with Optimal Utilization of Error Estimates of Data Values. *Environmetrics*, 5(2):111–126, 1994.

[268] Liviu Panait and Sean Luke. A Comparison of Two Competitive Fitness Functions. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 503–511. Morgan Kaufmann Publishers Inc., 2002.

[269] BK Panigrahi, V Ravikumar Pandi, Sanjoy Das, and Swagatam Das. Multiobjective Fuzzy Dominance Based Bacterial Foraging Algorithm to Solve Economic Emission Dispatch Problem. *Energy*, 35(12):4761–4770, 2010.

[270] Jan Paredis. Co-evolutionary Constraint Satisfaction. *Parallel Problem Solving from Nature - PPSN III*, pages 46–55, 1994.

[271] Jan Paredis. Steps Towards Co-evolutionary Classification Neural Networks. In *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 102–108, 1994.

[272] Jan Paredis. Coevolutionary Computation. *Artificial life*, 2(4):355–375, 1995.

[273] Jan Paredis. Coevolving Cellular Automata: Be Aware of the Red Queen!. In *ICGA*, pages 393–400. Citeseer, 1997.

[274] V Paul Pauca, Farial Shahnaz, Michael W Berry, and Robert J Plemmons. Text Mining Using Non-negative Matrix Factorizations. In *SDM*, volume 4, pages 452–456. SIAM, 2004.

[275] Tomasz P. Pawlak. *Competent Algorithms for Geometric Semantic Genetic Programming*. PhD thesis, Poznan University of Technology, Poznan, Poland, 21 September 2015. URL http://www.cs.put.poznan.pl/tpawlak/link/?PhD.

[276] Tomasz P. Pawlak and Krzysztof Krawiec. Semantic Geometric Initialization. In Malcolm I. Heywood, James McDermott, Mauro Castelli, Ernesto Costa, and Kevin Sim, editors, *EuroGP 2016: Proceedings of the 19th European Conference on Genetic Programming*, volume 9594 of *LNCS*, pages 261–277, Porto, Portugal, 30 March–1 April 2016. Springer Verlag. doi: 10.1007/978-3-319-30668-1_17.

[277] Tomasz P. Pawlak, Bartosz Wieloch, and Krzysztof Krawiec. Semantic Backpropagation for Designing Search Operators in Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 19(3):326–340, June 2015. ISSN 1089-778X. doi: 10.1109/TEVC.2014.2321259.

[278] Tomasz P. Pawlak, Bartosz Wieloch, and Krzysztof Krawiec. Review and Comparative Analysis of Geometric Semantic Crossovers. *Genetic Programming and Evolvable Machines*, 16(3):351–386, September 2015. ISSN 1389-2576. doi: 10.1007/s10710-014-9239-8.

[279] Michael Pazzani and Dennis Kibler. The Utility of Knowledge in Inductive Learning. *Machine learning*, 9(1):57–94, 1992.

[280] Dan Pelleg, Andrew W Moore, et al. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *ICML*, pages 727–734, 2000.

[281] Justyna Petke, William B. Langdon, and Mark Harman. Applying Genetic Improvement to MiniSAT. In Guenther Ruhe and Yuanyuan Zhang, editors, *Symposium on Search-Based Software Engineering*, volume 8084 of *Lecture Notes in Computer Science*, pages 257–262, Leningrad, August 2013. Springer. doi: 10.1007/978-3-642-39742-4_21. URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/Petke_2013_SSBSE.pdf. Short Papers.

[282] Justyna Petke, Mark Harman, William B. Langdon, and Westley Weimer. Using Genetic Improvement and Code Transplants to Specialise a C++ Program to a Problem Class. In Miguel Nicolau, Krzysztof Krawiec, Malcolm I. Heywood, Mauro Castelli, Pablo Garcia-Sanchez, Juan J. Merelo, Victor M. Rivas Santos, and Kevin Sim, editors, *17th European Conference on Genetic Programming*, volume 8599 of *LNCS*, pages 137–149, Granada, Spain, 23-25 April 2014. Springer. doi: 10.1007/978-3-662-44303-3_12. URL http://www0.cs.ucl.ac.uk/staff/J.Petke/papers/Petke_2014_EuroGP.pdf.

[283] Gordon Plotkin. Automatic Methods of Inductive Inference. 1972.

[284] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. URL http://www.gp-field-guide.org.uk. (With contributions by J. R. Koza).

[285] J.B. Pollack, A.D. Blair, and M. Land. Coevolution of a Backgammon Player. In *Artificial Life V: Proc. of the Fifth Int. Workshop on the Synthesis and Simulation of Living Systems*, pages 92–98. Cambridge, MA: The MIT Press, 1997.

[286] Jordan B Pollack and Alan D Blair. Co-evolution in the Successful Learning of Backgammon Strategy. *Machine learning*, 32(3):225–240, 1998.

[287] E. Popovici and K. De Jong. Understanding Competitive Co-evolutionary Dynamics Via Fitness Landscapes. In *Artificial Multiagent Symposium. Part of the 2004 AAAI Fall Symposium on Artificial Intelligence*, 2004.

[288] E. Popovici and K. De Jong. Relationships between Internal and External Metrics in Co-evolution. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, pages 2800–2807. IEEE, 2005.

[289] E. Popovici, A. Bucci, R.P. Wiegand, and E.D. De Jong. Coevolutionary Principles. *Handbook of Natural Computing*, 2010.

[290] Elena Popovici. Bridging Supervised Learning and Test-based Co-optimization. *Journal of Machine Learning Research*, 18(38):1–39, 2017.

[291] Elena Popovici and Kenneth De Jong. Monotonicity Versus Performance in Co-optimization. In *Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms*, pages 151–170. ACM, 2009.

[292] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. Coevolutionary Principles. In *Handbook of Natural Computing*, pages 987–1033. Springer, 2012.

[293] Mitchell A Potter and Kenneth A De Jong. Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents. *Evolutionary computation*, 8(1):1–29, 2000.

[294] William Poundstone. *Prisoner's Dilemma: John Von Neuman, Game Theory, and the Puzzle of the Bomb*. Doubleday, New York, 1992.

[295] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. ISBN 0471619779.

[296] Ingo Rechenberg. Evolutionsstrategie Optimierung Technischer Systeme Nach Prinzipien Der Biologischen Evolution. 1973.

[297] Craig Reynolds. Competition, Coevolution and the Game of Tag. In R. A. Brooks and P. Maes, editors, *Artificial Life IV, Proceedings of the fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 59–69, MIT, Cambridge, MA, USA, 1994. MIT Press.

[298] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to Recommender Systems Handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2011.

[299] Jon T Richardson, Mark R Palmer, Gunar E Liepins, and Mike R Hilliard. Some Guidelines for Genetic Algorithms with Penalty Functions. In *Proceedings of the 3rd international conference on genetic algorithms*, pages 191–197. Morgan Kaufmann Publishers Inc., 1989.

[300] Jose L. Risco-Martin, David Atienza, J. Manuel Colmenar, and Oscar Garnica. A Parallel Evolutionary Algorithm to Optimize Dynamic Memory Managers in Embedded Systems. *Parallel Computing*, 36(10-11):572–590, 2010. ISSN 0167-8191. doi: 10.1016/j.parco.2010.07.001. URL http://www.sciencedirect.com/science/article/B6V12-50J9GPR-1/2/e049c72f4c9e284bd1c2bdbf7c09f3aa. Parallel Architectures and Bioinspired Algorithms.

[301] Jorma Rissanen. Modeling by Shortest Data Description. *Automatica*, 14(5):465–471, 1978.

[302] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A Survey of Multi-objective Sequential Decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[303] Christopher D Rosin and Richard K Belew. Methods for Competitive Co-evolution: Finding Opponents Worth Beating. In *ICGA*, pages 373–381, 1995.

[304] Christopher D Rosin and Richard K Belew. New Methods for Competitive Coevolution. *Evolutionary computation*, 5(1):1–29, 1997.

[305] Duncan Roy. Learning and the Theory of Games. *Journal of Theoretical Biology*, 204(3):409–414, 2000.

[306] Thomas Runarsson and Simon Lucas. Preference Learning for Move Prediction and Evaluation Function Approximation in Othello. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(3):300–313, 2014. ISSN 1943-068X. doi: 10.1109/TCIAIG.2014.2307272.

[307] Thomas Philip Runarsson and Simon M Lucas. Coevolution Versus Self-play Temporal Difference Learning for Acquiring Position Evaluation in Small-board Go. *IEEE Transactions on Evolutionary Computation*, 9(6):628–640, 2005.

[308] Carl Runge. Über Empirische Funktionen Und Die Interpolation Zwischen äquidistanten Ordinaten. *Zeitschrift für Mathematik und Physik*, 46(224-243):20, 1901.

[309] Spyridon Samothrakis, Simon Lucas, ThomasPhilip Runarsson, and David Robles. Coevolving Game-playing Agents: Measuring Performance and Intransitivities. *IEEE Transactions on Evolutionary Computation*, 17(2):213–226, 2013.

[310] Arthur L Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[311] Tatsuya Sato and Takaya Arita. Competitive Co-evolutionary Algorithms Can Solve Function Optimization Problems. *Artificial Life and Robotics*, 14(3):440–443, 2009.

[312] Dhish Kumar Saxena, Joao A Duro, Ashutosh Tiwari, Kalyanmoy Deb, and Qingfu Zhang. Objective Reduction in Many-objective Optimization: Linear and Nonlinear Algorithms. *Evolutionary Computation, IEEE Transactions on*, 17(1):77–99, 2013.

[313] J David Schaffer. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Proceedings of the First International Conference on Genetic Algorithms and Their Applications, 1985*. Lawrence Erlbaum Associates. Inc., Publishers, 1985.

[314] James David Schaffer. *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms (artificial Intelligence, Optimization, Adaptation, Pattern Recognition)*. PhD thesis, Nashville, TN, USA, 1984. AAI8522492.

[315] Robert E Schapire. The Strength of Weak Learnability. *Machine learning*, 5(2):197–227, 1990.

[316] Michael Schmidt and Hod Lipson. Age-fitness Pareto Optimization. In Rick Riolo, Trent McConaghy, and Ekaterina Vladislavleva, editors, *Genetic Programming Theory and Practice VIII*, volume 8 of *Genetic and Evolutionary Computation*, chapter 8, pages 129–146. Springer, Ann Arbor, USA, 20-22 May 2010. URL http://www.springer.com/computer/ai/book/978-1-4419-7746-5.

[317] Michael D. Schmidt and Hod Lipson. Coevolution of Fitness Predictors. *IEEE Transactions on Evolutionary Computation*, 12(6):736–749, December 2008. ISSN 1089-778X. doi: 10.1109/TEVC.2008.919006.

[318] Eric Schulte, Stephanie Forrest, and Westley Weimer. Automated Program Repair through the Evolution of Assembly Code. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, pages 313–316, Antwerp, 20-24 September 2010. ACM. doi: 10.1145/1858996.1859059. URL http://www.cs.unm.edu/~eschulte/data/ase2010-asm-preprint.pdf.

[319] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., 1981.

[320] Travis C. Service and Daniel R. Tauritz. Co-optimization Algorithms. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 387–388, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-130-9. doi: 10.1145/1389095.1389166.

[321] C. E. Shannon. A Mathematical Theory of Communication. *Bell Systems Technical Journal*, 27: 623–656, 1948.

[322] Ehud Y Shapiro. *Inductive Inference of Theories from Facts*. Yale University, Department of Computer Science, 1981.

[323] Ehud Y Shapiro. *Algorithmic Program Debugging*. MIT press, 1983.

[324] J Shapiro. Does Data-model Co-evolution Improve Generalization Performance of Evolving Learners? In *Parallel Problem Solving from Nature - PPSN V*, pages 540–549. Springer, 1998.

[325] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587): 484–489, 2016.

[326] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the Game of Go without Human Knowledge. *Nature*, 550(7676):354, 2017.

[327] Karl Sims. Evolving 3d Morphology and Behavior by Competition. *Artificial life*, 1(4):353–372, 1994.

[328] Hemant Kumar Singh, Amitay Isaacs, and Tapabrata Ray. A Pareto Corner Search Evolutionary Algorithm and Dimensionality Reduction in Many-objective Optimization Problems. *Evolutionary Computation, IEEE Transactions on*, 15(4):539–556, 2011.

[329] Moshe Sipper and Eytan Ruppin. Co-evolving Architectures for Cellular Machines. *Physica D: Nonlinear Phenomena*, 99(4):428–441, 1997.

[330] Burrhus F. Skinner. *The behavior of organisms: An experimental analysis*. Appleton-Century, 1938.

[331] Robert E Smith, Stephanie Forrest, and Alan S Perelson. Searching for Diverse, Cooperative Populations with Genetic Algorithms. *Evolutionary computation*, 1(2):127–149, 1993.

[332] Lee Spector. *Automatic Quantum Computer Programming: A Genetic Programming Approach*, volume 7. Springer Science & Business Media, 2006.

[333] Lee Spector. Assessment of Problem Modality by Differential Performance of Lexicase Selection in Genetic Programming: A Preliminary Report. In Kent McClymont and Ed Keedwell, editors, *1st workshop on Understanding Problems (GECCO-UP)*, pages 401–408, Philadelphia, Pennsylvania, USA, 7-11 July 2012. ACM. doi: 10.1145/2330784.2330846. URL http://hampshire.edu/lspector/pubs/wk09p4-spector.pdf.

[334] Lee Spector and Alan Robinson. Genetic Programming and Autoconstructive Evolution with the Push Programming Language. *Genetic Programming and Evolvable Machines*, 3(1):7–40, March 2002. ISSN 1389-2576. doi: 10.1023/A:1014538503543. URL http://hampshire.edu/lspector/pubs/push-gpem-final.pdf.

[335] Lee Spector, David M. Clark, Ian Lindsay, Bradford Barr, and Jon Klein. Genetic Programming for Finite Algebras. In Maarten Keijzer, Giuliano Antoniol, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Nikolaus Hansen, John H. Holmes, Gregory S. Hornby, Daniel Howard, James Kennedy, Sanjeev Kumar, Fernando G. Lobo, Julian Francis Miller, Jason Moore, Frank Neumann, Martin Pelikan, Jordan Pollack, Kumara Sastry, Kenneth Stanley, Adrian Stoica, El-Ghazali Talbi, and Ingo Wegener, editors, *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1291–1298, Atlanta, GA, USA, 12-16 July 2008. ACM. doi: 10.1145/1389095.1389343. URL http://www.cs.bham.ac.uk/~wbl/biblio/gecco2008/docs/p1291.pdf.

[336] Matej Sprogar. A Study of GP's Division Operators for Symbolic Regression. In *Seventh International Conference on Machine Learning and Applications, ICMLA '08*, pages 286–291, La Jolla, San Diego, USA, 11-13 December 2008. IEEE. doi: 10.1109/ICMLA.2008.84.

[337] Kenneth O Stanley and Risto Miikkulainen. Competitive Coevolution through Evolutionary Complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004.

[338] M. Szubert, W. Jaśkowski, and K. Krawiec. Learning Board Evaluation Function for Othello by Hybridizing Coevolution with Temporal Difference Learning. *Control and Cybernetics*, 40(3), 2011.

[339] Marcin Szubert. *Coevolutionary Shaping for Reinforcement Learning*. PhD thesis, Poznan University of Technology, 2014. URL http://www.cs.put.poznan.pl/mszubert/pub/phdthesis.pdf.

[340] Marcin Szubert, Wojciech Jaśkowski, and Krzysztof Krawiec. Coevolutionary Temporal Difference Learning for Othello. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 104–111. IEEE, 2009.

[341] Marcin Szubert, Wojciech Jaśkowski, and Krzysztof Krawiec. On Scalability, Generalization, and Hybridization of Coevolutionary Learning: A Case Study for Othello. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(3):214–226, 2013. doi: 10.1109/TCIAIG.2013.2258919. URL http://www.cs.put.poznan.pl/mszubert/pub/szubert2013tciaig.pdf.

[342] Marcin Szubert, Wojciech Jaśkowski, Paweł Liskowski, and Krzysztof Krawiec. The Role of Behavioral Diversity and Difficulty of Opponents in Coevolving Game-playing Agents. In *EvoApplications 2015*, volume 9028 of *Lecture Notes in Computer Science*, pages 394–405. Springer, 2015.

[343] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the Gap to Human-level Performance in Face Verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[344] Julian Togelius, Peter Burrow, and Simon M Lucas. Multi-population Competitive Co-evolution of Car Racing Controllers. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 4043–4050. IEEE, 2007.

[345] Marco Tomassini, Leonardo Vanneschi, Philippe Collard, and Manuel Clergue. A Study of Fitness Distance Correlation As a Difficulty Measure in Genetic Programming. *Evolutionary Computation*, 13(2):213–239, Summer 2005. ISSN 1063-6560. doi: 10.1162/1063656054088549.

[346] Alan Mathison Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[347] Joseba Urzelai, Dario Floreano, Marco Dorigo, and Marco Colombetti. Incremental Robot Shaping. *Connection Science*, 10(3-4):341–360, 1998.

[348] Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O'Neill, R. I. McKay, and Edgar Galvan-Lopez. Semantically-based Crossover in Genetic Programming: Application to Real-valued Symbolic Regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, June 2011. ISSN 1389-2576. doi: 10.1007/s10710-010-9121-2.

[349] David Allen Van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD thesis, Wright Patterson AFB, OH, USA, 1999. AAI9928483.

[350] Leonardo Vanneschi, Mauro Castelli, and Sara Silva. A Survey of Semantic Methods in Genetic Programming. *Genetic Programming and Evolvable Machines*, 15(2):195–214, June 2014. ISSN 1389-2576. doi: 10.1007/s10710-013-9210-0.

[351] Stephen A Vavasis. On the Complexity of Nonnegative Matrix Factorization. *SIAM Journal on Optimization*, 20(3):1364–1377, 2009.

[352] S. Viswanathan and J.B. Pollack. On the Coevolutionary Construction of Learnable Gradients. In *Proceedings of the 2005 AAAI Fall Symposium on Coevolutionary and Coadaptive Systems. AAAI Press*, 2005.

[353] R.A. Watson and J.B. Pollack. Coevolutionary Dynamics in a Minimal Substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 702–709. Morgan Kaufmann, 2001.

[354] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. Automatically Finding Patches Using Genetic Programming. In Stephen Fickas, editor, *International Conference on Software Engineering (ICSE) 2009*, pages 364–374, Vancouver, May 2009. doi: 10.1109/ICSE.2009.5070536. URL http://www.cs.unm.edu/~tnguyen/papers/genprog.pdf.

[355] David R. White, Andrea Arcuri, and John A. Clark. Evolutionary Improvement of Programs. *IEEE Transactions on Evolutionary Computation*, 15(4):515–538, August 2011. ISSN 1089-778X. doi: 10.1109/TEVC.2010.2083669.

[356] Shimon Whiteson, Nate Kohl, Risto Miikkulainen, and Peter Stone. Evolving Soccer Keepaway Players through Task Decomposition. *Machine Learning*, 59(1):5–30, 2005.

[357] Paweł Widera, Jonathan M. Garibaldi, and Natalio Krasnogor. Evolutionary Design of the Energy Function for Protein Structure Prediction. In Andy Tyrrell, editor, *2009 IEEE Congress on Evolutionary Computation*, pages 1305–1312, Trondheim, Norway, 18-21 May 2009. IEEE Computational Intelligence Society, IEEE Press. doi: 10.1109/CEC.2009.4983095.

[358] R Paul Wiegand. *An Analysis of Cooperative Coevolutionary Algorithms*. PhD thesis, PhD thesis, George Mason University, Fairfax, Virginia, 2003.

[359] R Paul Wiegand, William C Liles, and Kenneth A De Jong. An Empirical Analysis of Collaboration Methods in Cooperative Coevolutionary Algorithms. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 1235–1242. Morgan Kaufmann Publishers Inc., 2001.

[360] Josh L. Wilkerson and Daniel Tauritz. Coevolutionary Automated Software Correction. In Juergen Branke, Martin Pelikan, Enrique Alba, Dirk V. Arnold, Josh Bongard, Anthony Brabazon, Juergen Branke, Martin V. Butz, Jeff Clune, Myra Cohen, Kalyanmoy Deb, Andries P Engelbrecht, Natalio Krasnogor, Julian F. Miller, Michael O'Neill, Kumara Sastry, Dirk Thierens, Jano van Hemert, Leonardo Vanneschi, and Carsten Witt, editors, *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1391–1392, Portland, Oregon, USA, 7-11 July 2010. ACM. doi: 10.1145/1830483.1830739.

[361] Stephen Wolfram. *A New Kind of Science*, volume 5. Wolfram media Champaign, 2002.

[362] Stephen Wolfram et al. *Theory and Applications of Cellular Automata*, volume 1. World Scientific Singapore, 1986.

[363] David H Wolpert and William G Macready. No Free Lunch Theorems for Optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[364] David H Wolpert, William G Macready, et al. No Free Lunch Theorems for Search. Technical report, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[365] Dietmar Wolz and Pedro PB De Oliveira. Very Effective Evolutionary Techniques for Searching Cellular Automata Rule Spaces. *J. Cellular Automata*, 3(4):289–312, 2008.

[366] Sewall Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. *Proceedings of the Sixth International Congress on Genetics*, 1(6):356–366, 1932.

[367] Zhenyu Yang, Ke Tang, and Xin Yao. Large Scale Evolutionary Optimization Using Cooperative Coevolution. *Information Sciences*, 178(15):2985–2999, 2008.

[368] Taku Yoshioka, Shin Ishii, and Minoru Ito. Strategy Acquisition for the Game. *Strategy Acquisition for the Game "Othello" Based on Reinforcement Learning*, 82(12):1618–1626, 1999.

[369] Eckart Zitzler and Lothar Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE transactions on Evolutionary Computation*, 3(4): 257–271, 1999.

[370] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the Strength Pareto Evolutionary Algorithm. *TIK-report*, 103, 2001.

Poznan University of Technology
Faculty of Computing Science
Institute of Computing Science

Typeset using LaTeX in Latin Modern.

BibTeX:

```
@PHDTHESIS{LiskowskiPhd2018,
    author  = {Paweł Liskowski},
    title   = {Heuristic Algorithms for Discovery of Search Objectives in Test-based Problems},
    school  = {Poznan University of Technology},
    address = {Poznan, Poland},
    year    = {2018}
}
```