An Iterative Heuristic for State Justification in Sequential Automatic Test Pattern Generation

Aiman H. El-Maleh **Sadiq M. Sait** **Syed Z. Shazli**

Computer Engineering Department
King Fahd University of Petroleum and Minerals
Dhahran-31261, Saudi Arabia
e-mail: {aimane,sadiq,shazli}@ccse.kfupm.edu.sa

Abstract

State justification is one of the most time-consuming tasks in sequential Automatic Test Pattern Generation (ATPG). For states that are difficult to justify, deterministic algorithms take significant CPU time without much success most of the time. In this work, we adopt a hybrid approach for state justification. A new method based on Genetic Algorithms is proposed, in which we engineer state justification sequences vector by vector. The proposed method is compared with previous GA-based approaches. Significant improvements have been obtained for ISCAS benchmark circuits in terms of state coverage and CPU time.

1 Introduction

With today's technology, it is possible to build very large systems containing millions of transistors on a single integrated circuit. Designing such large and complex systems while meeting stringent cost and time-to-market constraints requires the use of computer-aided-design (CAD) tools. Increasing complexity of digital circuits in very large scale integration (VLSI) environment requires more efficient algorithms to support the operations performed by CAD tools [1]. Testing of integrated circuits is an important area which nowadays accounts for a significant percentage of the total design and production costs of ICs. For this reason, a large amount of research efforts have been invested in the last decade in the development of more efficient algorithms for the Automatic Test Pattern Generation (ATPG) for digital circuits [2]. In order to obtain acceptably high quality tests, design for testability (DFT) techniques are in use [3]. The first technique, called *full-scan design*, can be used to reduce the sequential test generation problem to a less difficult combinational test generation problem. In this technique, all memory elements are chained into shift registers so that they can be set to desired values and observed by shifting test patterns in and out. In large circuits however, this technique adversely affects the test application time as all the test vectors have to be scanned in and out of the flip-flops. Moreover, all of the memory elements may not be scanable in a given circuit [4]. In order to alleviate the test complexity, a second technique, called *partial-scan design*, is employed. This involves scanning a selected set of memory elements. Both these methods can add 10-20% hardware overhead. In case of a full scan design, a combinational test generator can be used to obtain tests. However, a sequential test generator is necessary in case of a partial scan or no-scan design [4]. The goal in this work is to use Genetic Algorithms (GAs) for generating sequences that will help the Automatic Test Pattern Generator (ATPG) in detecting more faults by reaching specific states. GAs are very well suited for optimization and search problems [5]. Several ATPGs have been reported which use genetic algorithms for simulation-based test generation. A good comparison is given in [3]. The main advantage of GA-based ATPGs, as compared to other approaches, is their ability to cover a larger search space in lower CPU time. This improves the fault coverage and makes these ATPGs capable of dealing with larger circuits. On the other hand, the main drawback consists in their inability to identify untestable faults [2]. Deterministic algorithms for combinational circuit test generation have proven to be more effective than genetic algorithms [6]. Higher fault coverages are obtained, and the execution time is significantly smaller. However, state justification using deterministic algorithms is a difficult problem, especially if design and tester constraints are considered [7]. In simulation-based ATPGs, the search proceeds in the forward direction only. Hence there are no backtracks

and state justification is easier as compared to deterministic ATPGs. In this work, a hybrid state justification approach is proposed, where both deterministic and genetic-based algorithms are employed. In evaluating this approach, we will conduct experiments in which a deterministic test generator will be employed initially. Untestable faults will be identified. The states which could not be reached in this phase, will be attempted in a genetic phase for state justification. Since Genetic Algorithms have been used successfully for combining useful portions of several candidate solutions to a given problem [5], we will try to genetically engineer sequences which justify the leftover states. In [8], Genetic Algorithms have been used for state justification. The length of the sequence was a function of the structural sequential depth of the circuit, where sequential depth is defined as the minimum number of flip-flops in a path between the primary inputs and the farthest gate. In case of feed-back loops, the structural sequential depth may not give a correct estimate of the number of vectors required for justifying a given state. Thus, if a state requires longer justification sequence, it will not be justified. The approach also does not take into account the quality of intermediate states reached and evaluates a chromosome only on the basis of the final state reached. In this work, we will use an incremental approach in which the length of the sequences will be dynamic. State justification sequences will be genetically engineered vector by vector. Even if some state remains unjustified after the genetic phase, the best sequence obtained in a given number of generations will be viewed as a partial solution. The determinisitc ATPG will be seeded with this sequence so that it may become able to reach previously unvisited regions of the search space. The remainder of this paper is organized as follows: Section 2 discusses application of genetic algorithms to sequential ATPG. In Section 3, genetic-based state justification is presented. Experimental results are given in Section 4. Section 5 concludes the paper.

2 Sequential ATPG and Genetic Algorithms

The goal of sequential circuit ATPG using the single stuck-at fault (SSF) model is to derive an input vector sequence such that, upon application of this input vector sequence, we obtain different output responses between the fault-free and faulty circuits. The SSF model is an abstraction of defects in a circuit which cause a single line connecting components to be permanently stuck either at logic 0 or logic 1 [9]. In this work, we assume the SSF model.

2.1 Complexity of sequential ATPG

Sequential ATPG is a much more complex process than combinational ATPG due to signal dependencies across multiple time frames [10]. It has been shown in [11] that the test generation problem for combinational circuits is NP-complete. The search space is of the order of 2^n, where 'n' is the number of inputs. For sequential circuit ATPG, the worst-case search space is 9^m, where m is the number of flip-flops. This exponential search space makes exhaustive ATPG search computationally impractical for large sequential circuits [4]. In the last years, one of the main goals of researchers was to develop effective algorithms for sequential circuit test pattern generation [12]. A lot of work has been done in the area of sequential circuit test generation using both deterministic and simulation based algorithms. The bottleneck in deterministic algorithms is line justification and backtracking. In simulation-based approaches, no backtracking is required but their quality in terms of fault coverage is generally lower [12]. It can however be improved with the help of GAs which are very well suited for optimization and search problems.

2.2 Using GA in Sequential ATPG

Genetic Algorithms work by analogy with Natural Selection as follows. First, a population pool of chromosomes is maintained. The chromosomes are strings of symbols or numbers. They might be as simple as strings of bits - the simplest type of strings possible. The chromosomes are also called the genotype (the coding of the solution). These chromosomes must be evaluated for fitness. Poor solutions are purged and small changes are made to existing solutions. The gene pool thus evolves steadily towards better solutions. In this work, we have used a Simple Genetic Algorithm as given in [13]. Several approaches to test generation using genetic algorithms have been proposed in the past [2], [6] - [8], [12], [14] - [21]. Fitness functions were used to guide the GA in finding a test vector or sequence that maximizes given objectives for a single fault or a group of faults. However, hard-to-test faults often could not be detected. GAs were used in different phases of the test generation process. In [15], [6] and [16], GA-based test generators were developed which used logic simulation for fitness evaluation. A fault simulator was used in [17] [18], and [19] for computing the fitness. The fitness functions were biased towards maximizing the number of faults detected and the number of fault effects propagated to the flip-flops. Several genetic parameters were experimented with in [17] and [20]. The fault coverage improved by more

than 40% for some benchmark circuits. These genetic-based ATPGs were however, not successful in propagating fault effects to the primary outputs. Moreover, they were unable to identify redundant faults. Hence, hybrid techniques were proposed in [7], [8], [12] [14] and [21].

3 Genetic-based State Justification

State justification is the most difficult task in sequential ATPG. Storing the complete state information for large circuits is impractical. Similarly, keeping a list of sequences capable of reaching each reachable state is also infeasible. State justification is therefore performed by using a GA. In [7] and [8], deterministic algorithms were used for fault excitation and propagation, and a GA was used for state justification. Sequences were evolved over several generations. The fitness of each individual was a measure of how closely the final state reached matched the desired state. A chromosome was represented by a sequence of vectors. Candidate sequences were simulated starting from the last state reached at the end of the previous test sequence. The objective was to engineer a test sequence that justified the required state. If a sequence was found which justified the required state, the sequence was added to the test set. In this work, we use GA for traversing from one state to another. Individual vectors are represented by chromosomes in the population and genetic operators are applied at individual bit positions. Deterministic ATPG is run for every target fault. First, the fault is activated and propagated to a primary output. Next, state justification is attempted. If the required state is justified by the deterministic ATPG, then the derived sequence is fault simulated and all detected faults are dropped from the faultlist. Otherwise, our GA-based algorithm attempts to justify the required state. A block diagram of the methodology is shown in Figure 1.

We have proposed an evolutionary meta-heuristic for the state justification phase. A flowchart of the heuristic used is shown in Figure 2.

3.1 Encoding of the chromosome

In our work, a chromosome represents a single vector. Each character of a chromosome in the population is mapped to a primary input. A binary encoding is used in this implementation.

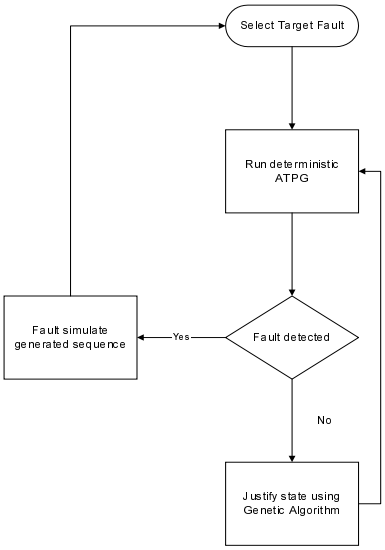


Figure 1: A block diagram of the methodology.

3.2 Fitness Function

Fitness function is the most important parameter of the GA. A solution is considered to be better than another if its fitness is higher. Each vector (chromosome) is logic simulated to give the state reached. This state is compared with all the flip-flop assignment values of the target state. The fitness $f(v_i)$ of a vector v_i is computed as follows:

$$f(v_i) = \frac{m(s_i,s_j)}{B(s_j)}$$

where s_i is the state reached by vector v_i, s_j is the target state and $m(s_i, s_j)$ are the number of matching specified bits in s_i and s_j. B(s_j) gives the number of specified bits in s_j (i.e. those which are not 'x').

3.3 Crossover and Mutation

One-point uniform crossover as mentioned in [22] has been used in this work. In one-point uniform crossover, an integer position is randomly selected within a chromosome. Each of the two parents are divided into two parts at this random cut point. The offspring is then generated by catenating the segment of one parent to the left of the cut point with the segment of the second parent to the right of the cut point. *Mutation* produces incremental changes in the offspring by randomly changing values of some genes. In this work, mutation corresponds to flipping a randomly selected bit.

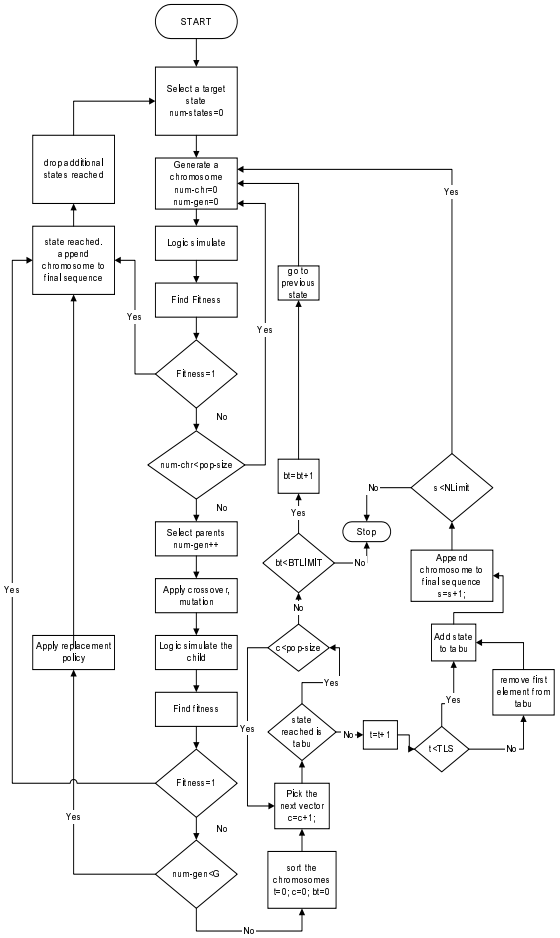


Figure 2: A flowchart of the algorithm used.

3.4 Forming a new generation

A generation is an iteration of GA where individuals in the current population are selected for crossover and offsprings are created. Due to the addition of offsprings, the size of population increases. In order to keep the number of members in a population fixed, a constant number of individuals are selected from this set for the new generation. The new population thus consists of both members from the initial generation and the offsprings created. In this work, we have used a one-change strategy as described below.

3.4.1 (n+1) selection strategy

In this strategy, we change one chromosome in every generation. One crossover is performed in every generation. If the child is more fit than the worst member of the previous generation, it is introduced into the population. Hence, we select the best n-1 members from a population of n, and the worst member gets replaced if its fitness is less than the fitness of the offspring.

3.5 Traversing from a state to a state

The algorithm is run for a fixed number of generations. If the state reached is the desired state, the algorithm stops and picks the next state from the list. However, if the algorithm is unable to reach the desired state, it picks the best chromosome found until then and adds it to the test set. Since the state reached is nearer in terms of the Hamming distance to the desired state, it is probable that it will help the ATPG in reaching the required search space and detecting the associated fault. The following parameters are used to guide the search.

3.5.1 Tabu List Size

To prevent the algorithm from visiting recently visited states, we propose a Tabu List containing the last visited states. The length of this list is a user-defined parameter. On reaching a state, the algorithm looks into the Tabu list. If the state reached is present, the next fit vector is chosen and its fitness is evaluated.

3.5.2 Backtrack limit

When all the chromosomes in the population are unable to reach a new state, (a state which is not in the Tabu List), we move to a previously visited state. This is termed as *backtracking*. We impose an upper limit on this parameter and the algorithm stops searching for a state when this parameter exceeds.

3.5.3 Nlimit parameter

The algorithm traverses *at least* Nlimit number of states before it gives up the search for the desired state. If the fitness of the currently visited state is less than the average fitness of the last Nlimit states, the algorithm stops further searching of the desired state; otherwise the search is continued.

3.6 Removing the reached states from the list of desired states

Once a sequence is generated by the algorithm, we compare the states reached by the sequence with the list of desired states. All the desired states reached by the sequence are removed. This prevents us from searching again for those states which we have already reached while searching for some other target state.

4 Experimental Results and Discussion

In this work, we have compared our state justification technique in which we use GA for traversing from a state to a state, with the one proposed in [7][8]. In [8], GA has been used in state justification and sequences are genetically engineered. GA has been applied on a sequence of vectors as opposed to individual vectors in our case. We have used five ISCAS89 benchmark circuits [23] and four re-timed circuits given in [10] for which which HITEC [24] requires very large CPU times. A list of target states was obtained for hard-to-detect faults in each of the circuits. We have experimented with several parameters and found that in general, a population size of 16, a generation limit of 400, backtrack limit of 10 and tabu list size of 15 gave the best results. Better results were obtained for an Nlimit value which was 1.5 times the number of flipflops present in the circuit. A roulette wheel selection scheme as given in [6] gave the best results. The weakest chromosome in the population was replaced by a new chromosome in every generation. Hence, the average and best fitness of the population monotonically increased in every generation as shown in Figure 3 for one of the reached target states. One-point crossover was used with a probability of 1 and mutation rate was kept at 0.01. In Figure 4, we show the state traversal for one of the states that has been reached by the algorithm. It can be seen that we progress towards better states in terms of the hamming distance as the algorithm runs for more iterations. Less fit states are reached if we we are unable to reach a better state because of the Tabu restriction. Moreover, we move towards the best state among all alternatives, even if that state is worse than the current state. This helps in avoiding the local minima. The example is for one of the target states of s1488 circuit.

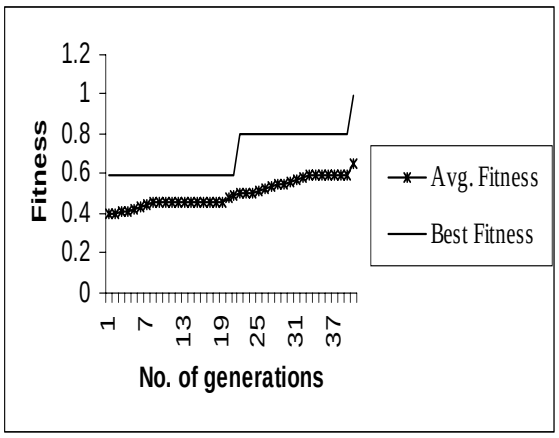


Figure 3: Average and best fitness vs. number of generations.

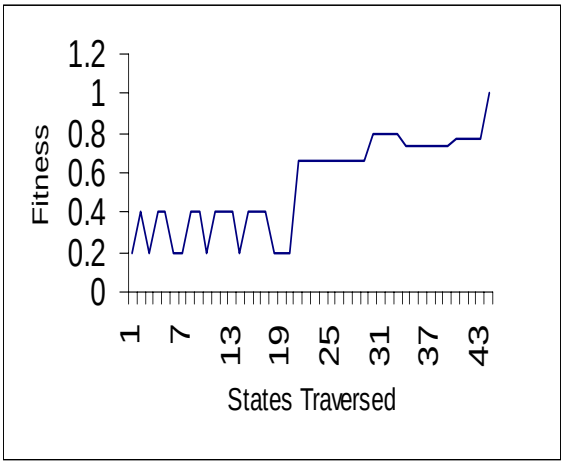


Figure 4: State traversed vs the fitness of reached states for a target state of s1488 circuit.

The parameters proposed in [8] were 32 chromosomes and 8 generations. The number of vectors in each chromosome was 4 times the sequential depth of the circuit.

To compute the fitness of chromosomes, we have used the logic simulator of HOPE [25]. The experiments were run on SUN ULTRA 10 stations and the results were obtained as shown in Table 1.

The first column in the table shows the circuit name. In the second and third columns, the number of flipflops (FFs) and the number of target states respectively is given for each circuit. The states reached and CPU time obtained by our algorithm are mentioned in the next two columns. For comparison purposes we ran the algorithm proposed in [8] for several number of generations and the results are shown in the next columns.

It can be observed from the results that the number of desired states reached by our technique are more than those reached by the technique used in [8] for all the circuits. Furthermore, our proposed technique reached a higher number of states than [8] in 8 out of 9 cases even when the latter was run for greater amount of CPU time.

5 Conclusion

In this work, we have proposed a new state justification technique based on GA which engineers the sequence

			our approach		approach in [8]			approach in [8]		
Name	# of FF	Target states	states reached	time(sec)	gens	states reached	time(sec)	gens	states reached	time(sec)
s1423	74	135	61	335	8	50	2743	50	61	3953
s3271	116	45	20	1229	8	15	1664	100	18	2390
s3384	183	102	56	8124	8	31	3794	200	42	16411
s5378	179	524	113	29274	8	45	3133	100	48	225160
s6669	239	32	30	1664	8	23	1701	50	24	2289
scfRjisdre	20	267	48	803	8	25	501	100	31	5196
s832jcsrre	31	57	8	139	8	7	120	100	7	2170
s510Rjcsrre	30	114	16	163	8	12	61	100	13	504
s510Rjosrre	32	114	16	181	8	9	62	100	13	583

Table 1: Comparison of the two techniques

vector by vector. This is in contrast to previous approaches where GA is applied to the whole sequence. The drawback of previous approaches lies in their inability to justify hard-to-reach states because of fixed-length sequences. Moreover, they do not take into account the quality of intermediate states reached and evaluate a chromosome only on the basis of the final state reached. We propose dynamic length sequences in this work and the fitness measure takes into account all the states reached by the sequence. The approach has been compared with previous approaches and improvements in reached states and CPU time have been demonstrated.

Acknowledgment: The authors thank King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for support.

References

[1] Aiman El-Maleh, Mark Kassab, and Janusz Rajski. A Fast simulation-based learning technique for real sequential circuits. In *Design Automation Conference*, 1998.

[2] F.Corno, P.Prinetto, M.Rebaudengo, and Sonza Reorda. A Parallel Genetic Algorithm for aoutomatic generation of test sequences for digital circuits. In *International Conf. on High Performance Computing and Networking, Belgium*, April 1996.

[3] Y.C.Kim and K.K.Saluja. Sequential test generators: past, present and future. *INTEGRATION, the VLSI journal*, 26:41–54, 1998.

[4] M.H.Konijnenburg, J.T. van der Linden, and A.J. van de Goor. Sequential test generation with advanced illegal state search. In *International Test Conference*, 1997.

[5] Srinivas M. and L.M.Patnaik. Genetic Algorithms: A Survey. *IEEE Computer Magazine*, pages 17–26, June 1994.

[6] E.M.Rudnick, J.Holm, D.G.Saab, and J.H.Patel. Application of Simple Genetic Algorithm to sequential circuit test generation. In *European Design and Test Conference*, pages 40–45, February 1994.

[7] Elizabeth M. Rudnick and Janak H. Patel. State justification using Genetic Algorithms in sequential circuit test generation. *A survey report from CRHC Univ. of Illinois, Urbana*, January 1996.

[8] M.S.Hsiao, E.M.Rudnick, and J.H.Patel. Application of genetically engineered finite-state-machine sequences to sequential circuit ATPG. *IEEE Transactions on CAD of Integrated circuits and systems*, 17:239–254, March 1998.

[9] Xinghao Chen and M.L.Bushnell. Sequential circuit test generation using dynamic justification equivalence. *Journal of Electronic Testing*, 8:9–33, 1996.

[10] T.E.Marchok, Aiman El-Maleh, W.Maly, and J.Rajski. A complexity analysis of sequential ATPG. *IEEE Transactions on CAD of Integrated circuits and systems*, 15:1409–1423, November 1998.

[11] P.H.Ibarra and S.K.Sahni. Polynomially complete fault detection problems. *IEEE Transactions on Computing*, 24:242–249, 1975.

[12] Michael S. Hsiao. Use of Genetic Algorithms for testing sequential circuits. *Ph.D. Dissertation, UIUC*, December 1997.

[13] D.E.Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[14] F.Corno, P.Prinetto, M.Rebaudengo, Sonza Reorda, and M.Violante. Exploiting logic simulation to improve simulation-based sequential ATPG. In *Sixth IEEE Asian Test Symposium, Arta, Japan*, November 1997.

[15] D.G.Saab, Y.G.Saab, and J.A.Abraham. CRIS: A test cultivation program for sequential VLSI circuits. In *International Conf. on Computer-aided Design*, pages 216–219, November 1992.

[16] F.Corno, P.Prinetto, M.Rebaudengo, and Sonza Reorda. GATTO: A genetic algorithm for automatic test pattern generation for large synchronous sequential circuits. *IEEE Transactions on CAD of Integrated circuits and systems*, 15:991–1000, August 1996.

[17] M.S.Hsiao, E.M.Rudnick, and J.H.Patel. Alternating strategies for sequential circuit ATPG. In *European Design and Test Conference*, pages 368–374, March 1996.

[18] E.M.Rudnick, J.H.Patel, G.S.Greenstein, and T.M.Niermann. A Genetic algorithm framework for test generation. *IEEE Transactions on CAD of Integrated circuits and systems*, 16:1034–1044, September 1997.

[19] E.M.Rudnick, J.H.Patel, G.S.Greenstein, and T.M.Niermann. Sequential circuit test generation in a genetic algorithm framework. In *Design Automation Conference*, pages 698–704, June 1994.

[20] D.G.Saab, Y.G.Saab, and J.A.Abraham. Automatic test vector cultivation for sequential VLSI circuits using genetic algorithms. *IEEE Transactions on CAD*, 15:1278–1285, October 1996.

[21] M.S.Hsiao, E.M.Rudnick, and J.H.Patel. Sequential circuit test generation using dynamic state traversal. In *European Design and Test Conference*, pages 22–28, March 1997.

[22] Sadiq M.Sait and Habib Youssef. *Iterative Computer Algorithms wirh applications in Engineering: Solving combinatorial optimization problems.* IEEE Computer Society, 1999.

[23] F.Brglez, D.Bryan, and K.Kozminski. Combinational profiles of sequential benchmark circuits. *International Symposium on Circuits and Systems*, pages 1929–19347, 1989.

[24] T.M.Niermann and J.H.Patel. HITEC: A test generation package for sequential circuits. In *European Conference on Design Automation*, pages 214–218, 1991.

[25] H. K. Lee and D. S. Ha. HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15:1048–1058, September 1996.

Evolution Strategy with Neighborhood Attraction – A Robust Evolution Strategy

Jutta Huhse
University of Tübingen
Department of Computer Science - WSI-RA
Sand 1, D-72076 Tübingen, Germany
huhse@informatik.uni-tuebingen.de

Andreas Zell
University of Tübingen
Department of Computer Science - WSI-RA
Sand 1, D-72076 Tübingen, Germany
zell@informatik.uni-tuebingen.de

Abstract

The evolution strategy with neighborhood attraction (EN) is a new combination of self-organizing maps (SOM) and evolution strategies (ES). It adapts the neighborhood relationship known from SOM to ES individuals in order to concentrate them around the optimum of the problem.

In this paper, detailed investigations on the robustness of the EN were performed on a variety of well-known optimization problems. The behavior of the EN was compared to that of several other known variants of ES such as ES with mutative step control, ES with covariance matrix adaptation, differential evolution and others. In this test series, it was shown that EN is much more robust than the other ES variants.

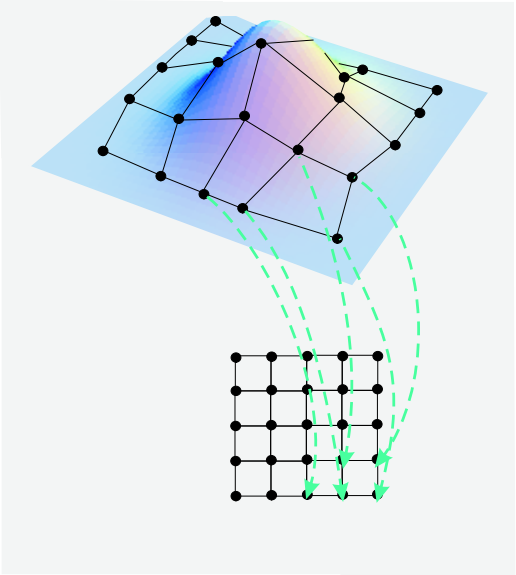


Figure 1: EN: Transfer of the SOM neighborhood onto ES individuals

1 INTRODUCTION

Evolution strategies with **N**eighborhood attraction (EN) are a combination of two different kinds of problem solvers: Evolution strategies (ES) and artificial neural networks, i.e. self-organizing maps (SOM), to be more precise.

ES were developed in the late 1960s by Rechenberg and Schwefel and later improved (see [Rechenberg, 1994], [Schwefel, 1995] and [Bäck et al., 1997]). Their main application is the optimization of real-valued multi-parameter problems. They directly use the information of the quality of a potential solution of the function to be optimized. ES work on a population P of potential solutions (individuals a) by manipulating these individuals with evolutionary operators.

A special class of neural networks - the self-organizing maps (SOM) - were developed in the 1980s by Kohonen [Kohonen, 1995]. The neurons of a SOM are organized in a neighborhood relationship, e.g. a two-dimensional grid. Learning takes place by adapting the weight vectors of the neurons (and thereby the neurons' positions in the problem space) according to a learning rule which incorporates the neighborhood relation defined among the neurons.

The idea behind EN is to transfer the neighborhood and the learning rule defined for SOM neurons onto the individuals of an ES (see fig. 1). Using this neighborhood concept in the new EN, better EN individuals can attract their worse neighbors and thus, the individuals will be concentrated around the optimum.

Previous benchmark tests were performed on a number of optimization tasks which could be solved

by EN as well as by many conventional ES (cf. [Huhse and Zell, 2000]). The main focus was on the convergence velocity of the EN, and it could be shown that – especially for small populations – the performance of the EN is equivalent to or even better than comparable conventional ES on those benchmark problems.

This paper investigates the robustness of the EN. The focus lies on difficult optimization tasks which often cause problems to conventional ES. A test bed of many difficult optimization tasks was set up, and the reliability of the EN is compared to several different variants of ES like ES with mutative step control, ES with covariance matrix adaptation, differential evolution, and others. The experiments show that the EN has an apparently better robustness than the other ES variants on most of the tested optimization tasks.

A short description of the EN is given in section 2. The optimization problems used as a test bed for our investigations are described in section 3. Section 4 shows the test series that were performed on EN and the ES variants, and the results are discussed in section 5. Detailed information on the test functions can be found in the appendix.

2 EVOLUTION STRATEGY WITH NEIGHBORHOOD ATTRACTION

The individuals which are unordered in conventional ES, have neighborhood relations in the EN. The neighborhood between the μ parent individuals is constituted by arranging them in an orthogonal, elastic grid. As known from SOM, each individual can be identified by its fixed grid position, and two individuals are neighbors if they are directly connected on the grid (see fig. 2, left).

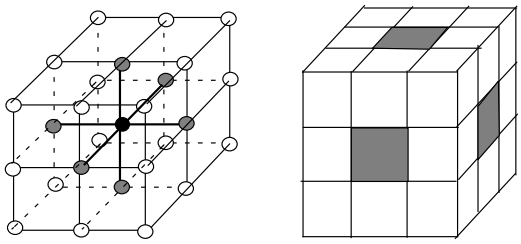


Figure 2: Left: Neighborhood grid between the parent individuals; all grey individuals are neighbors of the black individual.
Right: Division of the problem space into hyper-cubes; during initialization one individual is placed randomly into each hyper-cube

The dimension d_g of the grid depends on the dimension d_p of the problem space and on the number μ of the parent individuals (i.e. the number of the individuals constituting the grid). The grid is calculated in the following way: First, μ is divided into its prime factors; e.g. $\mu = 100 \Rightarrow 2 \cdot 2 \cdot 5 \cdot 5$; $n_f = 4$ is the number of the factors f_i. If the number n_f of the factors is smaller than the dimension of the problem, then the grid dimension is set to n_f. Otherwise, the smallest primes are multiplied until the number of factors is equal to the problem dimension. Thus, it holds: $d_g \leq d_p$. Inside the neighborhood grid the individuals are arranged according to the factorization. E.g. for $\mu = 100$ and a problem dimension $d_p = 10$, the grid dimension is $d_g = 4$ and $f_1 = 2$ individuals are in the first dimension, $f_2 = 2$ individuals in the second, $f_3 = 5$ individuals in the third and $f_4 = 5$ individuals in the fourth.

Because of the orthogonality of the grid the neighborhood is easily determined. The left and the right neighbor of one individual a_i can be determined independently for each dimension. E.g. for dimension $d = 1$ the two neighbors of a_i with the grid coordinates

$$\begin{aligned} a_i &= (a_0,\ a_1,\ \ a_2, \ldots, a_n) \quad \text{are} \\ a_{N_l} &= (a_0, a_1 - 1, a_2, \ldots, a_n) \quad \text{and} \\ a_{N_r} &= (a_0, a_1 + 1, a_2, \ldots, a_n). \end{aligned}$$

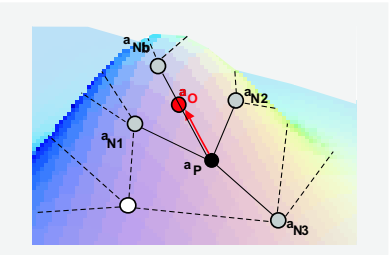


Figure 3: Neighborhood attraction in EN

In contrast to conventional ES and SOM, the initial values of the object variables of the EN individuals are not assigned randomly. Rather, the problem space is divided into equally sized hyper-cubes, each of them corresponding to one grid position (see fig. 2, right). The object variables of the associated individual are initialized with equally distributed random values within the ranges of its hyper-cube.

As is customary in ES, the EN individuals are evaluated using the fitness function.

The EN-specific evolutionary operator – the neighborhood attraction – manipulates the EN individuals ac-

cording to one learning step in a SOM. Every parent individual a_P is attracted to its best neighbor a_{Nb} and thus becomes the offspring a_O (see figure 3). The object variables $\vec{x}_O$ of the offspring are calculated according to equation 1 and the neighborhood relations are retained unchanged.

$$\vec{x}_O = \vec{x}_P + \delta \cdot (\vec{x}_{Nb} - \vec{x}_P) \tag{1}$$

Here, $\vec{x}_P$ is the object variables vector of the parent and $\vec{x}_{Nb}$ is its best neighbor. The parameter δ defines the strength of the attraction along the difference vector and $\vec{x}_O$ denotes the object variables of the offspring.

If the parent individual a_P is considered better than all its neighbors a_{Nj} ($j = 1 \ldots g$, g is the number of neighbors) a "simple conventional" mutation (referred to as *ES-mutation* here) is performed. λ offsprings are generated according to 2.

$$\begin{array}{rclcrcl} \vec{v}_{mut,l} & = & \vec{\mathcal{N}}(0,1) & & l & = & 1 \ldots \lambda \\ d_{min} & = & \min(\|\vec{x}_P - \vec{x}_{Nj}\|) & & j & = & 1 \ldots g \\ s_{eff} & = & \frac{1}{n} d_{min} & & & & \\ \vec{x}_{O_i} & = & \vec{x}_P + s_{eff} \cdot \vec{v}_{mut_i} & & i & = & 1 \ldots n \end{array} \tag{2}$$

n is the number of object variables.

The effective step size s_{eff} is determined by the reciprocal number of object variables [1] and by the distance d_{min} to the nearest neighbor. Thus, a mutation which jumps over a neighbor and an entanglement of the grid becomes less likely. During the contraction of the grid the effective step sizes decrease due to the influence of d_{min}.

Recombination is not explicitly used.

For details, please see [Huhse and Zell, 2000].

3 TEST FUNCTIONS

An extensive test bed of optimization tasks was constituted to permit thorough investigations on the robustness of the EN.

On the one hand, the functions used for previous test series [Huhse and Zell, 2000] were incorporated (f_1, f_2, f_6, f_9, f_{15}, f_{21}). These functions include uni-modal and multi-modal functions as well as symmetric and non-symmetric ones, and they were also used e.g. in [de Jong, 1975], [Bäck, 1992] and [Schwefel, 1995]. On most of these test functions almost all ES variants and also the EN converged.

On the other hand, special focus was set on very difficult test functions which are known to cause problems to many optimization tasks. E.g. f_5 (Shekel's foxholes) consists of a wide plateau with many steep and narrow holes embedded as local minima, where the individuals might get caught. f_{23} [Galar, 1991] consists of a plateau with one local maximum, connected to the global maximum by a single saddle. The varying dimensions of each test function are indicated in the appendix.

- f_1: Sphere model
- f_2: Generalized Rosenbrock's function
- f_5: Shekel's foxholes
- f_6: Schwefel's double sum
- f_9: Ackley's function
- f_{15}: Weighted sphere model
- f_{17}: Fletcher and Powell
- f_{18}: Shekel-5
- f_{19}: Shekel-7
- f_{20}: Shekel-10
- f_{21}: Griewangk
- f_{23}: Galar
- f_{24}: Kowalik

4 TEST SERIES

For each function of the test bed the EN was compared to the following ES variants:

- ES with uncorrelated self adaptation (uncorrelated)
- ES with covariance matrix adaptation (CMA) according to [Hansen and Ostermeier, 1996]
- ES with mutative step control (named MSR by [Rechenberg, 1994])
- ES with derandomized self adaptation (derand) [Ostermeier et al., 1993]
- ES without self adaptation (off)

[1] according to [Rechenberg, 1994], who proposes for his basic algorithm for a $(1^+, \lambda)$ -ES to make the length of the mutation vector independent of the number n of variables by generating the normally distributed vector elements with the mean zero and with the variance $\sigma = \frac{1}{\sqrt{n}}$

- ES with self adaptation adopted from differential evolution (diffevol) [Price and Storn, 1995]

The following parameter settings were used for the EN: Size of the individual grid $\mu = 100$, attraction factor $\delta = 0.0011$, and the number of offsprings per parent generated during *ES-mutation* $\lambda = 2$.

For all ES variants, a $(10, 100)$-strategy without recombination was used. These settings were chosen because they are known to be practicable for many optimization tasks. Furthermore $\lambda = 100$ corresponds to the grid size of the EN, which means, that the number of function evaluations which are calculated in one generation of EN corresponds to that of one generation of ES.

We developed a special EN simulation program to perform the test series. For the comparision tests we used *EvA*, a simulation program for Evolutionary Algorithms which was developed in the same group [Wakunda and Zell, 1997].

A simulation run was stopped when the convergence value was reached or when the algorithm stagnated. Focus was not on the number of function evaluations but on the best function value reached. Every run was repeated 30 times with different random seeds. Then the best fitness values reached were averaged out and the standard deviation was calculated. The graphical representations below show for the different strategies (abscissa) the average of the best fitness values and the standard deviation added to and subtracted from that average (ordinate). [2] For clarity, the optimum of each test function is plotted as a thin line.

Not all test series can be represented graphically here. For the test functions f_1, f_6, f_9, f_{15} almost all strategies were equally reliable in finding almost always the optimum. Only the differential evolution ES had some problems.

The more interesting results for the difficult functions which could not be solved by some strategies are shown below:

For function f_5, only EN was totally reliable (fig. 4). All other strategies were frequently trapped in one of the local optima.

For function f_{17}, most strategies achieved good results (fig. 5). Uncorrelated ES, CMA-ES and derandomized ES found the optimum with only small variance, the other variants had some more problems. Again, only EN showed no problems in the optimization.

[2] Only one standard deviation was calculated for better and worse results. Note: The subtracted standard deviation does not imply that there were results better than the optimum. It is only shown to facilitate the comparision of very similar results, like CMA, derand, and EN in fig. 5.

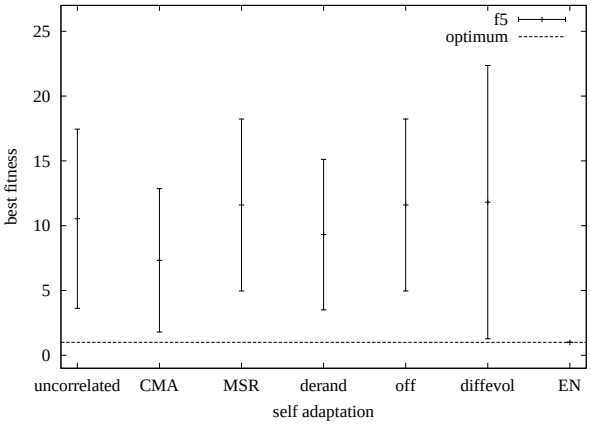


Figure 4: Function f_5, $\min(f_5(\vec{x})) = 0$

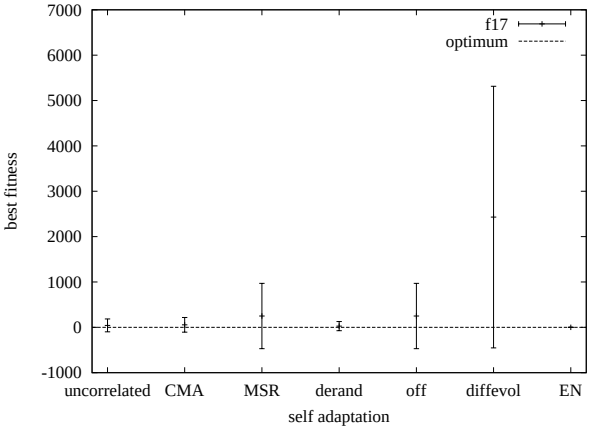


Figure 5: Function f_{17}, $\min(f_{17}(\vec{x})) = 0$

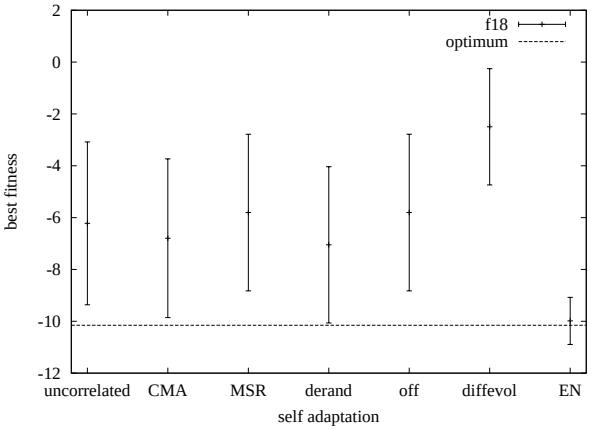


Figure 6: Function f_{18}, $\min(f_{18}(\vec{x})) = -10.1532$

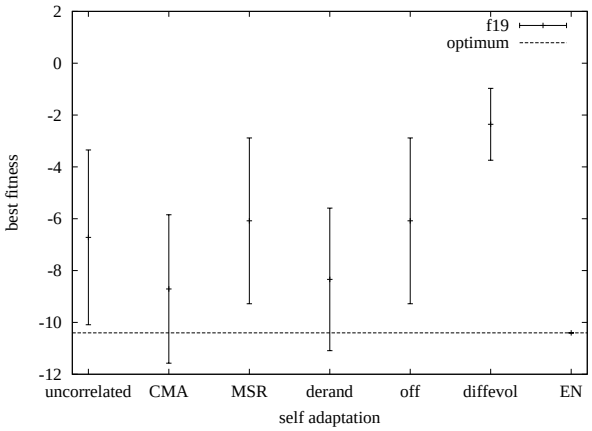


Figure 7: Function f_{19}, $\min(f_{19}(\vec{x})) = -10.4029$

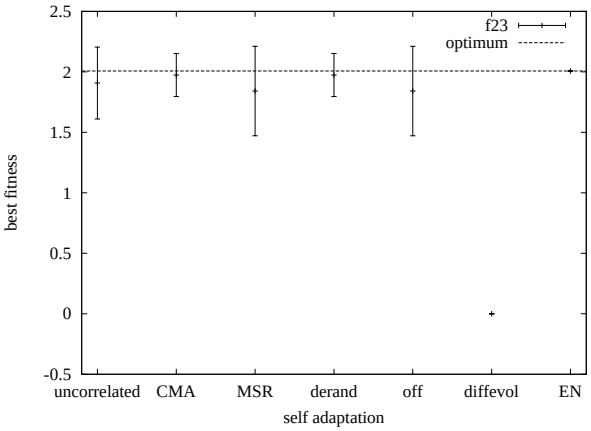


Figure 9: Function f_{23}, $\max(f_{23}(\vec{x})) = 2.00686$

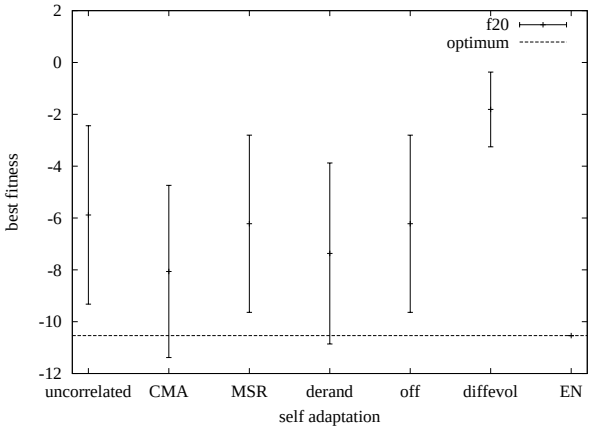


Figure 8: Function f_{20}, $\min(f_{20}(\vec{x})) = -10.5364$

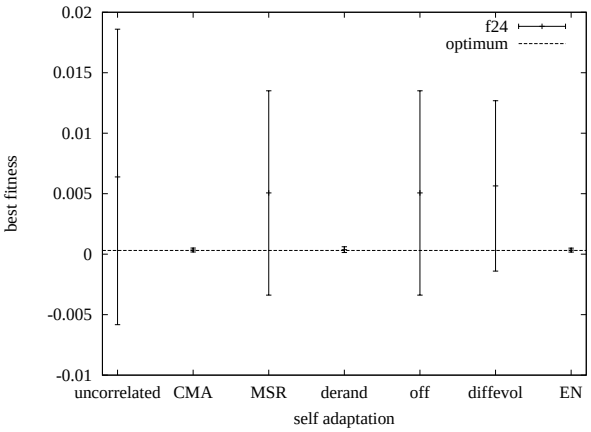


Figure 10: Function f_{24}, $\min(f_{24}(\vec{x})) = 0.0003075$

Also for f_{18}, EN is the most robust strategy (fig. 6). It stagnated only once, and compared to the other strategies, EN was clearly the best.

The test series for f_{19} and f_{20} led to similar results (fig. 7, fig. 8): EN was the only strategy which was able to always find the optimum. All other strategies stagnated repeatedly.

The same holds for the maximization problem f_{23} (fig. 9). EN always found the global optimum, while the other strategies often climbed on the local maximum, and one strategy (differential evolution ES) did not even leave the plateau from where the search started.

The test series with function f_{24} shows varying results (fig. 10). The best strategies are EN, CMA-ES, and derandomized ES. CMA-ES and EN are almost equal, derandomized ES performs a bit worse.

F_2 is the only test function where EN was outperformed by other strategies (fig. 11). While uncorrelated ES, MSR-ES, derandomized ES and even ES without self adaptation (off) converged always, EN had problems as well as CMA-ES and differential evo-

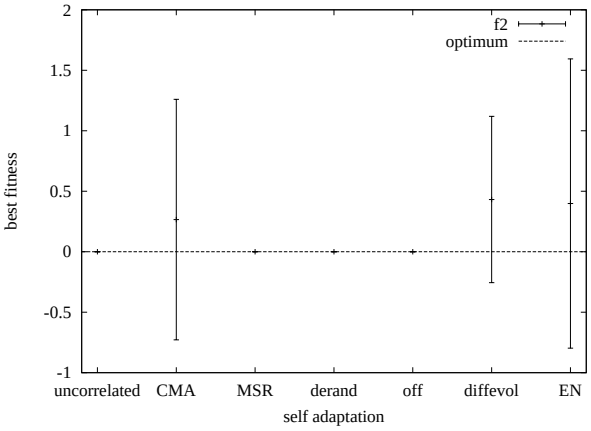


Figure 11: Function f_2, $\min(f_2(\vec{x})) = 0$

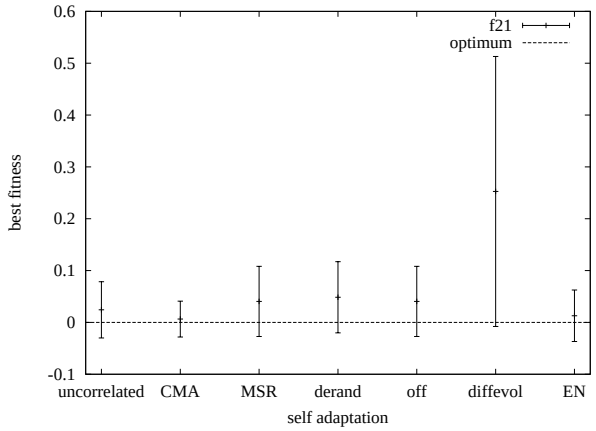


Figure 12: Function f_{21}, $\min(f_{21}(\vec{x})) = 0$

lution ES.

F_{21} is a quite difficult test function which could not be optimized reliably by any strategy (fig. 12). Comparing the averages of the best function values found, EN is the second best strategy after CMA-ES.

5 CONCLUSIONS

The robustness of the new EN strategy – Evolution strategy with Neighborhood attraction – was investigated in exhaustive test series using a large test bed of optimization tasks and many ES variations for comparision.

Only for one of the thirteen test series, some of the ES variants were more robust than EN, i.e. other strategies were able to optimize the test function more often than EN. For all test series, the EN was able to find the optimum in at least 90% of the test runs. For most of the test series, the EN converged always (for all seeds), and for many optimization tasks, EN was the only strategy that was able to converge always, while all other ES variants repeatedly stagnated in local optima.

It could be shown that the EN is much more robust than other ES-variants, especially for difficult, multimodal functions.

For further work it is conceivable to incorporate the mechanism of EN into other existing, elaborated ES to improve these. A combination of e.g. CMA and EN could be quite promising.

A TEST FUNCTIONS

A.1 f_1: Sphere model

[de Jong, 1975]

$f_1(\vec{x}) = \sum_{i=1}^{n} x_i^2$

$-5.12 \le x_i \le 5.12\,; \quad dim = 10$

$\min(f_1) = f_1(0,\ldots,0) = 0$

A.2 f_2: Generalized Rosenbrock's function

[de Jong, 1975]

$f_2(\vec{x}) = \sum_{i=1}^{n-1}(100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$

$-5.12 \le x_i \le 5.12\,; \quad dim = 10$

$\min(f_2) = f_2(1,\ldots,1) = 0$

A.3 f_5: Shekel's foxholes

[de Jong, 1975]

$\frac{1}{f_5(\vec{x})} = \frac{1}{K} + \sum_{j=1}^{25} \frac{1}{c_j + \sum_{i=1}^{2}(x_i - a_{ij})^6}$

$(a_{ij}) =$

$$\begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \cdots & 32 & 32 & 32 \end{pmatrix}$$

$K = 500 \quad f_5(a_{1j}, a_{2j}) \approx c_j = j$

$-65.536 \le x_i \le 65.536\,; \quad dim = 2$

$\min(f_5) = f_5(-32,-32) \approx 0.998004$

A.4 f_6: Schwefel's double sum

[Schwefel, 1981, Schwefel, 1995] (function 1.2)

$f_6(\vec{x}) = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2 = x^{\mathrm{T}}\mathbf{A}x$

$-65.536 \le x_i \le 65.536\,; \quad dim = 10$

$\min(f_6) = f_6(0,\ldots,0) = 0$

A.5 f_9: Ackley's function

[Ackley, 1987]

$f_9(\vec{x}) = -a \cdot \exp\left(-b\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right)$

$- \exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} \cos(c \cdot x_i)\right) + a + e$

$a = 20\,; \quad b = 0.2\,; \quad c = 2\pi$

$-32.768 \le x_i \le 32.768\,; \quad dim = 10$

$\min(f_9) = f_9(0,\ldots,0) = 0$

A.6 f_{15}: Weighted sphere model

[Schwefel, 1988]

$f_{15}(\vec{x}) = \sum_{i=1}^{n} i \cdot x_i^2$

$-5.12 \leq x_i \leq 5.12\ ; \quad dim = 10$

$\min(f_{15}) = f_{15}(0, \ldots, 0) = 0$

A.7 f_{17}: Fletcher and Powell

[Fletcher and Powell, 1963]

$f_{17}(\vec{x}) = \sum_{i=1}^{n} (A_i - B_i(x))^2$

$A_i = \sum_{j=1}^{n} (a_{ij} \sin \alpha_j + b_{ij} \cos \alpha_j)$

$B_i = \sum_{j=1}^{n} (a_{ij} \sin x_j + b_{ij} \cos x_j)$

$a_{ij}, b_{ij} \in [-100, 100]$ (equ. distr. randoms)

$\alpha_j \in [-\pi, \pi]$ (equ. distr. randoms)

$-\pi \leq x_i \leq \pi\ ; \quad dim = 4$

$\min(f_{17}) = f_{17}(\alpha_1, \alpha_2, \ldots, \alpha_n) = 0$

A.8 f_{18}: Shekel-5

[Törn and Žilinskas, 1989]

$f_{18}(\vec{x}) = -\sum_{i=1}^{m} \frac{1}{(\vec{x} - A(i))(\vec{x} - A(i))^T + c_i}$

$m = 5$

$0 \leq x_i \leq 10\ ; \quad dim = 4$

i	$A(i)$				c_i	$f_{18}(\vec{x} = A(i))$
1	4	4	4	4	0.1	-10.1532
2	1	1	1	1	0.2	-5.0552
3	8	8	8	8	0.2	-5.10076
4	6	6	6	6	0.4	-2.68284
5	3	7	3	7	0.4	-2.6304

$\min(f_{18}) = f_{18}(0, \ldots, 0) = -10.1532$

A.9 f_{19}: Shekel-7

[Törn and Žilinskas, 1989]

$f_{19}(\vec{x}) = -\sum_{i=1}^{m} \frac{1}{(\vec{x} - A(i))(\vec{x} - A(i))^T + c_i}$

$m = 7$

$0 \leq x_i \leq 10\ ; \quad dim = 4$

i	$A(i)$				c_i	$f_{19}(\vec{x} = A(i))$
1	4	4	4	4	0.1	-10.4028
2	1	1	1	1	0.2	-5.08767
3	8	8	8	8	0.2	-5.1288
4	6	6	6	6	0.4	-2.75186
5	3	7	3	7	0.4	-2.76589
6	2	9	2	9	0.6	-1.83708
7	5	5	3	3	0.3	-3.72275

$\min(f_{19}) = f_{19}(0, \ldots, 0) = -10.4029$

A.10 f_{20}: Shekel-10

[Törn and Žilinskas, 1989]

$f_{20}(\vec{x}) = -\sum_{i=1}^{m} \frac{1}{(\vec{x} - A(i))(\vec{x} - A(i))^T + c_i}$

$m = 10$

$0 \leq x_i \leq 10\ ; \quad dim = 4$

i	$A(i)$				c_i	$f_{20}(\vec{x} = A(i))$
1	4	4	4	4	0.1	-10.5363
2	1	1	1	1	0.2	-5.12847
3	8	8	8	8	0.2	-5.17562
4	6	6	6	6	0.4	-2.871
5	3	7	3	7	0.4	-2.80662
6	2	9	2	9	0.6	-1.85892
7	5	5	3	3	0.3	-3.83364
8	8	1	8	1	0.7	-1.67525
9	6	2	6	2	0.5	-2.42083
10	7	3.6	7	3.6	0.5	-2.42652

$\min(f_{20}) = f_{20}(0, \ldots, 0) = -10.5364$

A.11 f_{21}: Griewangk

[Törn and Žilinskas, 1989]

$f_{21}(\vec{x}) = 1 + \frac{1}{d} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$

$d = 200$

$-100.0 \leq x_i \leq 100.0\ ; \quad dim = 10$

$\min(f_{21}) = f_{21}(0, \ldots, 0) = 0$

A.12 f_{23}: Galar

[Galar, 1991]

$f_{23}(\vec{x}) = (\exp(-5x_1^2) + 2\exp(-5(1 - x_1)^2))$

$\cdot \exp\left(-5 \sum_{i=2}^{n} x_i^2\right)$

$-5.0 \leq x_i \leq 5.0\ ; \quad dim = 10$

$\min(f_{23}) = f_{23}(0.9965, 0, \ldots, 0) \approx 2.00686$

A.13 f_{24}: Kowalik

[Schwefel, 1977, Schwefel, 1995]

$$f_{24}(\vec{x}) = \sum_{i=1}^{11} \left(a_i - \frac{x_1(b_i^2+b_i x_2)}{b_i^2+b_i x_3+x_4} \right)^2$$

$$-5.0 \leq x_i \leq 5.0\,; \quad dim = 4$$

i	a_i	b_i^{-1}
1	0.1957	0.25
2	0.1947	0.5
3	0.1735	1
4	0.1600	2
5	0.0844	4
6	0.0627	6
7	0.0456	8
8	0.0342	10
9	0.0323	12
10	0.0235	14
11	0.0246	16

$$\min(f_{24}) \approx f_{24}(0.1928, 0.1908, 0.1231, 0.1358)$$

$$\approx 0.0003075$$

References

[Ackley, 1987] Ackley, D. H. (1987). *A connectionist machine for genetic hillclimbing*. Kluwer Academic Publishers, Boston.

[Bäck, 1992] Bäck, T. (1992). A user's guide to genesys 1.0. Technical report, University of Dortmund, Department of Computer Science

[Bäck et al., 1997] Bäck, T., Hammel, U., and Schwefel, H.-P. (1997). Evolutionary computation. Comments on the history and current state. In *IEEE Transactions on Evolutionary Computation*, volume 1, pages 3–17.

[de Jong, 1975] de Jong, K. (1975). An analysis of the behaviour of a class of genetic adaptive systems. Master's thesis, University of Michigan.

[Fletcher and Powell, 1963] Fletcher, R. and Powell, M. J. D. (1963). A rapidly convergent descent method for minimization. *Comp. J. 6*, pages 163–168.

[Galar, 1991] Galar, R. (1991). Simulation of local evolutionary dynamics of small populations. *Biological Cybernetics*, 65:37–45.

[Hansen and Ostermeier, 1996] Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adatation. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC '96)*, pages 312–317, Nagoya, Japan.

[Huhse and Zell, 2000] Huhse, J. and Zell, A. (2000). Evolution strategy with neighborhood attraction. In *Proceedings of the Second ICSC Symposium on Neural Computation (NC'2000)*, pages 363–369.

[Kohonen, 1995] Kohonen, T. (1995). *Self-Organizing Maps*. Springer Verlag, Berlin.

[Ostermeier et al., 1993] Ostermeier, A., Gawelczyk, A., and Hansen, N. (1993). A derandomized approach to self adaptation of evolution strategies. Technical report, Technische Universität Berlin.

[Price and Storn, 1995] Price, K. and Storn, R. (1995). Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute (ICSI), University of California at Berkeley.

[Rechenberg, 1994] Rechenberg, I. (1994). *Evolutionsstrategie '94*. frommann-holzboog, Stuttgart.

[Schwefel, 1977] Schwefel, H.-P. (1977). *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, volume 26 of *Interdisciplinary systems research*. Birkhäuser, Basel.

[Schwefel, 1981] Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. Wiley, Chichester.

[Schwefel, 1988] Schwefel, H.-P. (1988). Evolutionary learning optimum–seeking on parallel computer architectures. In Sydow, A., Tzafestas, S. G., and Vichnevetsky, R., editors, *Proceedings of the International Symposium on Systems Analysis and Simulation 1988, I: Theory and Foundations*, pages 217–225. Akademie der Wissenschaften der DDR, Akademie-Verlag, Berlin.

[Schwefel, 1995] Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*. John Wiley and Sons, New York.

[Törn and Žilinskas, 1989] Törn, A. and Žilinskas, A. (1989). *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer, Berlin.

[Wakunda and Zell, 1997] Wakunda, J. and Zell, A. (1997). Eva - a tool for optimization with evolutionary algorithms. In *Proceedings of the 23rd EUROMICRO Conference*, Budapest, Hungary.

On the Utility of Populations in Evolutionary Algorithms

Thomas Jansen
FB 4, LS 2
Univ. Dortmund
44221 Dortmund, Germany
jansen@ls2.cs.uni-dortmund.de

Ingo Wegener
FB 4, LS 2
Univ. Dortmund
44221 Dortmund, Germany
wegener@ls2.cs.uni-dortmund.de

Abstract

Evolutionary algorithms (EAs) are population-based search heuristics often used for function optimization. Typically they employ selection, crossover, and mutation as search operators. It is known that EAs are outperformed by simple hill-climbers in some cases. Thus, it may be asked whether the use of a population and crossover is at all advantageous. In this paper it is rigorously proven that the use of a population instead of just a single individual can be an advantage of its own even without making use of crossover. This establishes by example the advantage of EAs compared to (random) hill-climbers on appropriate objective functions. Moreover, we describe one particular situation where intuitively a population should outperform a single individual and present a formal proof justifying this intuition.

1 INTRODUCTION

The class of evolutionary algorithms comprises genetic algorithms, evolution strategies, evolutionary programming, and genetic programming. All these general and robust randomized search heuristics are applied to many different tasks. Though there is a huge amount of successful applications, from a theoretical point of view our understanding of evolutionary algorithms is still dissatisfactory. Here, we concentrate on the task of optimization and consider the maximization of pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$.

One common theoretical approach is the investigation of very simple evolutionary algorithms. This can lead to the investigation of the $(1+1)$ EA in the extreme case, an evolutionary algorithm using a population of size 1, no crossover, and deterministic plus-selection as in evolution strategies (Mühlenbein (1992), Droste, Jansen, and Wegener (1998), Garnier, Kallel, and Schoenauer (1999)). This algorithm, that can be considered to be a kind of randomized hill-climber, is interesting not only from a theoretical point of view, but also is of practical interest. It is known (Juels and Wattenberg (1995), Mitchell (1995)) that this simple algorithm can outperform quite sophisticated evolutionary algorithms. This holds even on functions where one would expect the use of crossover to be of great benefit (Mitchell, Forrest, and Holland (1992), Mitchell, Holland, and Forrest (1994)). This motivates that we mistrust intuitions and look for confirmation by rigorous proofs. Thus, we want to find at least one example where we can prove that a population based evolutionary algorithm can outperform the $(1+1)$ EA. Since in practice it is common to restart an evolutionary algorithm after some time, we want the $(1+1)$ EA to be outperformed even if arbitrary restart strategies are allowed. The evolutionary algorithm should in some sense be "standard": We do not want a specialized algorithm this is tailored for just this one example. Therefore, we want to apply standard techniques, only. However, since we want to prove that the use of a population alone can be advantageous, we do not allow the use of any crossover operator. Moreover, the same mutation operator has to be applied in both algorithms, since we do not want any performance differences to stem from different mutation operators. We are interested in a comparison of the computation time. Note, that the number of generations alone is in general not an appropriate measure. If only the number of generations is taken into account, the use of a large population can only be advantageous. Such a measurement is obviously cheating. Under such unrealistic assumptions it is easy to outperform search heuristics based on a single individual. Here, we choose the number of function evaluations as an appropriate measure that is

easy to define and to measure. For the algorithms considered it will be obvious that the number of function evaluations is a realistic measure for the computation time.

For continuous search space, especially for the sphere function, some results are known. Beyer (1993,1995a,1995b) shows that as far as local performance on the sphere function is concerned the use of populations cannot help. In the case of a noisy environment things are different. Arnold and Beyer (2000) argue how the use of populations can be helpful in a noisy environment and increase the efficiency of the search. Jansen and Wegener (1999) prove that a genetic algorithm can by far outperform the $(1+1)$ EA. This result relies heavily on the use of uniform crossover. Therefore, it remains open whether the use of a population alone, even without crossover, can be advantageous. Here, we give a rigorous proof that this can be the case. In the next section we give precise definitions of the algorithms used. Furthermore, we explain an intuitive idea why the use of a population may be advantageous and define a family of functions in accordance to that idea. In Section 3, we consider one special member of that family of functions and prove that a population-based algorithm with appropriately chosen population size can clearly outperform the (1+1) EA. Based upon the methods from this first result we derive a much stronger statement. We show that a population-based algorithm with a population that is not too small optimizes a broad family of functions in polynomial expected time while the $(1+1)$ EA has superpolynomial expected run time. Finally, in Section 4 we summarize and discuss open questions.

2 DEFINITIONS

The (1+1) EA is perhaps the most simple EA, using a population size of just 1 and mutation, only. The selection mechanism used is the plus-selection known from evolution strategies. This simple algorithm is subject of intense research, see for example Mühlenbein (1992), Rudolph (1997), Droste, Jansen, and Wegener (1998), Garnier, Kallel, and Schoenauer (1999). We assume that we want to maximize some objective function $f\colon \{0,1\}^n \to \mathbb{R}$.

Algorithm 1 (The $(1+1)$ EA).

1. Choose $x \in \{0,1\}^n$ uniformly at random.
2. Create y by flipping each bit in x independently with probability $1/n$.
3. If $f(y) \geq f(x)$, then set $x := y$.
4. Continue at 2.

Since the (1+1) EA uses a kind of degenerated population of size 1 and accepts only strings with at least equal function value it can be regarded as a kind of random mutation hill-climber. Note, however, that since in each step it generates any $x' \in \{0,1\}^n$ as child y with positive probability, it cannot get stuck in any local optimum. This distinguishes the (1+1) EA from other hill-climbing algorithms.

We want to compare the efficiency of the (1+1) EA with that of an evolutionary algorithm that uses a real population but no crossover. We want to prove that such an EA can indeed be superior to the simple (1+1) EA. On the one hand, we would like to have an EA for this purpose that is a kind of "standard EA" except for neglecting crossover. On the other hand, it is immediately obvious that advantages due to the use of a population can only occur if the population does not converge too rapidly. Therefore, we would like to apply some mechanism that helps us to maintain at least some minimal degree of diversity. There are numerous more or less complicated mechanisms to do so, see Bäck, Fogel, and Michalewicz (1997) for an overview. A very simple mechanism is avoidance of duplicates as suggested and investigated by Ronald (1998). An even simpler and computationally less expensive mechanism is avoidance of replications as used by Jansen and Wegener (1999). Avoidance of replications ensures that every child is subject to a mutation flipping at least 1 bit or to crossover. We will see that this weak and computationally cheap mechanism is sufficient to maintain an acceptable degree of diversity.

The EA we use is a kind of elitist steady-state GA, using fitness-proportional selection for reproduction, bit-wise mutations with probability $1/n$ (just as the (1+1) EA), no crossover, reverse proportional selection for deletion, and deletion after insertion. We use population size N and leave the concrete choice of N open for the moment.

Algorithm 2 (The Evolutionary Algorithm).

1. For $i := 1$ To N Do
   Choose $x_i \in \{0,1\}^n$ uniformly at random.
2. Choose $y \in \{x_1, x_2, \ldots, x_N\}$ such that $\mathrm{Prob}(y = x_i) = f(x_i) / \sum_{j=1}^{N} f(x_j)$.
3. Repeat
4. Create x_{N+1} by flipping each bit in y independently with probability $1/n$.
5. Until $x_{N+1} \neq y$.
6. Sort $\{x_1, x_2, \ldots, x_{N+1}\}$ such that $f(x_1) \geq f(x_2) \geq \cdots \geq f(x_{N+1})$ holds.

7. Choose $z \in \{x_2, x_3, \ldots, x_{N+1}\}$ such that $\text{Prob}(z = x_i) = \frac{f(x_1)+f(x_{N+1})-f(x_i)}{\sum_{j=2}^{N+1}(f(x_1)+f(x_{N+1})-f(x_j))}$ and remove it.
8. Continue at 2.

One may argue that it is not appropriate to call Algorithm 2 a GA since it does not use crossover. Using mutation as the only variation operator seems to make this algorithm to be similar to an evolution strategy or evolutionary programming. In order to avoid this debate we denote Algorithm 2 as evolutionary algorithm.

Our intuitive idea for the origin of an advantage for the population-based EA compared to the (1+1) EA is the following (Rowe 2000). Assume we have a function that mainly consists of an easy to follow path to a local optimum that has second best function value. Then a direct mutation to a global optimum is needed. The expected waiting time for such a mutation can be quite large depending on the Hamming distance between the local and a global optimum. Since Algorithm 2 uses a population and the quite weak proportional selection, it has a good chance to "diffuse" the population around the local optimum. Therefore, some members of the population have a smaller Hamming distance to a global optimum what can lead to a quicker finding of this global optimum. We define a class of objective functions with such properties and prove for some members of this class that they have the desired properties such that the (1+1) EA is outperformed by the population-based evolutionary algorithm. The function class is a modified (mainly scaled) version of JUMP_k as used by Jansen and Wegener (1999) to exemplify the utility of uniform crossover.

Definition 3. *The function* $\text{SJUMP}_{k,s}\colon \{0,1\}^n \to \mathbb{R}$ (*short for* SCALEDJUMP) *is defined for any* $n \in \mathbb{N}$, $s \in \mathbb{N} \setminus \{1\}$, $k \in \{1, 2, \ldots, n\}$ *by*

$$\text{SJUMP}_{k,s}(x) := \begin{cases} s^{\|x\|_1} & \text{if } (\|x\|_1 \leq n-k) \vee (\|x\|_1 = n) \\ s^{n-k} + n - k - \|x\|_1 & \text{otherwise,} \end{cases}$$

for each $x \in \{0,1\}^n$, *where* $\|x\|_1$ *denotes the number of ones in* x.

3 RESULTS

The results for the (1+1) EA follow more or less directly from the investigations of the (1+1) EA on JUMP_k by Jansen and Wegener (1999). For the sake of completeness we present full proofs here, too.

Theorem 4. *The expected run time of the* (1+1) *EA on* $\text{SJUMP}_{k,s}\colon \{0,1\}^n \to \mathbb{R}$ *for* $k > 1$, $k = O(\log n)$, *and* $s \in \mathbb{N} \setminus \{1\}$ *is* $\Theta\left(n^k\right)$. *For any* $t \in \mathbb{N}$, *the probability that the* (1+1) *EA optimizes* $\text{SJUMP}_{k,s}$ *within* t *steps is bounded above by* $O\left(t/n^k\right)$.

Proof. We begin with an upper bound on the expected run time. Consider levels of strings

$$l_i := \{x \in \{0,1\}^n \mid \|x\|_1 = i\}$$

for $i \in \{0, 1, \ldots, n\}$. Note, that for all i and all $x, y \in l_i$ we have $\text{SJUMP}_{k,s}(x) = \text{SJUMP}_{k,s}(y)$ since $\text{SJUMP}_{k,s}$ is a symmetric function. Furthermore, for all $i \neq j$ and all $x \in l_i$ and all $y \in l_j$ we have $\text{SJUMP}_{k,s}(x) \neq \text{SJUMP}_{k,s}(y)$. Thus, once the (1 + 1) EA leaves some level l_i (that is we have $x \in l_i$, $y \notin l_i$ and $\text{SJUMP}_{k,s}(x) \leq \text{SJUMP}_{k,s}(y)$) no string from l_i can ever become the current string. Let p_i denote the probability that in one generation the (1 + 1) EA leaves l_i. Clearly, then $\sum_{i=0}^{n-1} 1/p_i$ is an upper bound on the expected run time. For $i < n - k$ it is sufficient to mutate exactly one of the $n - i$ bits with value 0. For $i > n - k$ it is sufficient to mutate exactly one of the i bits with value 1. This yields $p_i \geq (\min\{i, n-i\}/n)(1-1/n)^{n-1}$ for $i \neq n-k$. For $i = n - k$ we have $p_i = \left(1/n^k\right)(1-1/n)^{n-k}$, since the only way to increase the function value is to mutate exactly the k bits with value 0 leading to the unique global optimum. Thus, we have

$$\left(1 - \frac{1}{n}\right)^{1-n} \cdot \left(\left(2 \sum_{\substack{0 \leq i \leq n/2 \\ i \neq n-k}} \frac{n}{n-i}\right) + n^k\right) = \text{O}\left(n^k\right)$$

as upper bound on the expected run time. This upper bound holds for all values of $k \in \{1, 2, \ldots, n\}$.

Now, we prove the second statement. Note, that the lower bound on the expected run time follows from this statement. It is essential that we have $k > 1$ and $k = \text{O}(\log n)$, here. We denote the current string of the (1 + 1) EA in the t-th step by x_t. After random initialization we have $\|x_0\|_1 \leq n - k$ with probability $1 - e^{-\Omega(n)}$ due to Chernoff bounds (Motwani and Rhagavan (1995). Once we have $\|x_t\|_1 = n - k$, we have $\|x'_t\|_1 \in \{n - k, n\}$ for all $t' \geq t$. For all x with $\|x\|_1 < n - k$ the probability to reach the unique global optimum from x via a direct mutation is $o\left(1/n^k\right)$. Thus, we only have to deal with strings x with $\|x\|_1 > n - k$. In order to reach the unique global optimum via a direct mutation from such a string the following two events are necessary. First, such a string has to be reached. Second, the global optimum has to be reached from this string before some string x' with $\|x'\|_1 = n - k$ becomes current

string of the $(1+1)$ EA. For all $y \in \{0,1\}^n$ with $\|y\|_1 = n-m$, we have that y is reached from any string x with $\|x\|_1 \leq n-k$ with probability $\mathrm{O}\left(k^m/n^{k+1-m}\right)$. This holds since the Hamming distance from x to y is bounded below by $k+1-m$. Assume, that we have $x_t = y$ at some point of time. Let t^* be the first point of time where the current string is either the global optimum or contains exactly $n-k$ ones, i. e. $t^* = \min\{t' \mid t' > t \wedge \|x_{t'}\|_1 \in \{n-k, k\}\}$. For all t' with $t \leq t' < t^*$ we have $\|x_{t'}\|_1 \leq n-m$. Thus, the probability to reach the global optimum in one step is always bounded above by $\mathrm{O}\left(1/n^m\right)$. The probability to increase the number of zeros by 1 in one step is bounded below by $((n-k)/n) \cdot (1-1/n)^{n-1}$. We consider only steps t' with $x_{t'} \neq x_{t'-1}$. We see that the probability to reach the global optimum in such a step is still bounded above by $\mathrm{O}\left(1/n^m\right)$. Note, that after at most $k-m$ such steps we have reached x_{t^*}. Thus, the probability that x_{t^*} is the unique global optimum is bounded above by $\mathrm{O}\left((k-m)/n^m\right) = \mathrm{O}\left(k/n^m\right)$. Therefore, we have that the probability to reach the global optimum from y is bounded above by $\mathrm{O}\left(k^{m+1}/n^{k+1}\right) = \mathrm{O}\left((k/n)^{k+1}\right) = o\left(1/n^k\right)$ for all such strings y. This completes the proof. □

Obviously, the simple $(1+1)$ EA is able to optimize $\textsc{SJump}_{k,s}$ quite efficiently, at least for small k. However, we see that arbitrary restart-strategies cannot decrease significantly the expected run time. Note, that the size of s does not matter for this algorithm. This is different for Algorithm 2, which uses a fitness-sensitive selection mechanism, namely proportional selection. We concentrate on the special case $\textsc{SJump}_{2,n^2}$, first. This is useful, since in this simpler case it is easier to develop the methods and insights that will yield the result in more general cases, too. Here, however, no too big advantage is possible. It is important to see that arbitrary restart mechanisms cannot lead to a substantial speed-up of the $(1+1)$ EA. This is due to the upper bound of $\mathrm{O}\left(t/n^k\right)$ for the success probability after t steps.

Theorem 5. *The expected run time of Algorithm 2 with population size $N = \lfloor\sqrt{n}\rfloor$ on the function $\textsc{SJump}_{2,n^2}$ is bounded above by $O\left(n^{3/2}\right)$.*

Before we prove the result for the population-based evolutionary algorithm we consider two events that are of special interest and both have to do with selection. First, we are interested in a lower bound on the probability to select some member of the population with a certain function value as parent.

Lemma 6. *Consider Algorithm 2 on the function $\textsc{SJump}_{k,s}$. Assume that x is a member of the current population. Let f_1 and f_2 denote the function value of the current best and current second best members of the population. Consider the selection in line 3 of the algorithm.*

a) The probability to select some member y of the current population with $\textsc{SJump}_{k,s}(y) = \textsc{SJump}_{k,s}(x)$ is bounded below by $\textsc{SJump}_{k,s}(x)/(Nf_1)$.

b) The probability to select some member y of the current population with $\textsc{SJump}_{k,s}(y) = f_1$ is bounded below by $1-(f_2N)/f_1$.

The other aspect of selection is selection for deletion in line 7. Here, we are interested in an upper bound on the probability to select some member of the extended population with a certain function value for deletion.

Lemma 7. *Consider Algorithm 2 on the function $\textsc{SJump}_{k,s}$, in particular, consider the selection in line 7 of the algorithm. We make the following assumptions about the current population:*

- *The population contains $N+1$ strings.*
- *The current best member of the population contains exactly $n-k$ ones.*
- *There is exactly one member x of the population which contains exactly j ones, where we have $n-k < j < n$.*

a) The probability to select x is bounded above by $2/N$.

b) If there is at least one member of the population with less than $n-k$ ones, then the probability to select x is bounded above by $1/(s+N)$.

We omit the proofs of Lemma 6 and Lemma 7. A technical report with full proofs is available.

Proof of Theorem 5. We use a population size of $N = \lfloor\sqrt{n}\rfloor$ and assume for the sake of notational simplicity that we have $\lfloor\sqrt{n}\rfloor = \sqrt{n}$. The generalization to other cases is trivial. We describe a kind of typical run that ends when the unique global optimum becomes member of the current population for the first time. We divide that run into two disjoint phases and estimate the expected length of each phase. The first phase begins after random initialization and ends when a string with $n-2$ ones or the unique global optimum becomes member of the current population for the first time. The second phase begins after the first phase and ends when the unique global optimum becomes member of

the current population for the first time. Due to the elitism employed in Algorithm 2 we always have at least one string with $n-2$ ones in the population in the second phase.

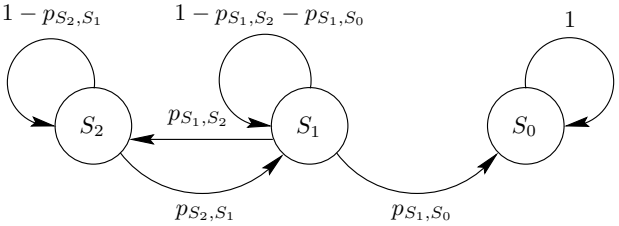
The expected length of the first phase is $\mathrm{O}(n \log n)$: We estimate the probability to select the current best member of the population as parent in line 3. Either the whole population consists of N strings that all have current best function value, or we have $f_2/f_1 = \mathrm{O}\left(1/n^2\right)$. In the first case we obviously select such a parent with probability 1. In the latter case, we apply Lemma 6 b). Thus, we have $1 - \mathrm{O}\left(N/n^2\right) = 1 - \mathrm{O}\left(1/n^{3/2}\right)$ in any case as a lower bound for selecting the current best member of the current population for reproduction. Due to the elitistic replacement strategy the current best member of the population cannot be replaced. Therefore, we can pessimistically ignore steps where some other member of the population is chosen as parent. Now, consider the case that the chosen parent is the current best member of the population and contains i bits with value 1. Then, we have a probability of at least $((n-i)/n) \cdot (1-1/n)^{n-1}$ to create a child y with larger function value. Thus, the expected waiting time for improving the current best member of the population is bounded above by $\left(n^{3/2}/\left(n^{3/2}-1\right)\right) \cdot en/(n-i) = \mathrm{O}(n/(n-i))$ in this situation. We sum up these expected waiting times and get $\sum_{i=0}^{n-1} \mathrm{O}(n/(n-i)) = \mathrm{O}(n \log n)$ as upper bound on the expected length of the first phase. Note, that these considerations do not depend on the initial population.

In the second phase we are interested in three special events. First, we investigate the probability that we introduce a string with exactly $n-1$ ones in the population when there is no such string present in one generation. Second, we are interested in the probability to remove the only occurrence of a string with exactly $n-1$ ones without introducing the global optimum instead in one generation. Finally, we investigate the probability to create the global optimum in one generation given that at least one string with exactly $n-1$ ones is present in the current population.

We consider the current population and distinguish three different "states" it can be in. Note, that each of these three states does not represent one special population but a large number of different populations that all have some property in common. If no string in the current population has more than $n-2$ ones we say the EA is in state A. If the string with the maximal number of ones in the current population contains $n-1$ ones, we say the EA is in state B. Finally, if the global optimum is member of the current population we speak of state C. We want to derive an upper bound on the expected number of generations the EA needs to "reach state C", i. e. to have a population that satisfies the defining condition of state C.

We consider a Markov chain with three states S_0, S_1, and S_2. We want to bound the first hitting time of S_0 starting in S_2 from above. For $X, Y \in \{S_0, S_1, S_2\}$ we denote by $p_{X,Y}$ the transition probability for a state transition from X to Y.

We assume pessimistically that we have $p_{S_2,S_0} = 0$. This can obviously only enlarge the first hitting time of S_0. Since we are interested in the first hitting time of S_0, only, we can assume $p_{S_0,S_0} = 1$, so $p_{S_0,S_2} = p_{S_0,S_1} = 0$ follows. Due to our assumptions we have $p_{S_2,S_1} + p_{S_2,S_2} = 1$, so we can replace p_{S_2,S_2} by $1 - p_{S_2,S_1}$. In the same way we can replace p_{S_1,S_1} by $1 - p_{S_1,S_2} - p_{S_1,S_0}$. The resulting Markov chain can be visualized as follows.



Let the random number of transitions starting in S_2 leading to S_0 be denoted by T_{S_2}. Let the random number of transitions starting in S_1 leading to S_0 be denoted by T_{S_1}. Then we have $\mathrm{E}(T_{S_2}) = (1 - p_{S_2,S_1})(1 + \mathrm{E}(T_{S_2})) + p_{S_2,S_1}(1 + \mathrm{E}(T_{S_1}))$ which leads to $\mathrm{E}(T_{S_2}) = (1/p_{S_2,S_1}) + \mathrm{E}(T_{S_1})$.

For $\mathrm{E}(T_{S_1})$ we have $\mathrm{E}(T_{S_1}) = (1 - p_{S_1,S_0} - p_{S_1,S_2})(1 + \mathrm{E}(T_{S_1})) + p_{S_1,S_0} + p_{S_1,S_2}(1 + \mathrm{E}(T_{S_2}))$ leading to $\mathrm{E}(T_{S_1}) = (1/p_{S_1,S_0}) + (p_{S_1,S_2}/(p_{S_2,S_1} \cdot p_{S_1,S_0}))$.

We look for an upper bound on the expected absorption time and thus need an upper bound on p_{S_1,S_2} and lower bounds on p_{S_2,S_1} and p_{S_1,S_0}. In order to obtain meaningful results, we connect the described Markov chain to the evolutionary algorithm we consider. First of all, the defining conditions of the three different "states" of the EA are obviously not sufficient to describe unambiguously exactly one population. In fact, for each such "state" a lot of different populations satisfy the according defining condition. The probability to reach some population that meets condition B in one generation starting from a population that meets condition A depends on the specific population the EA starts in. The same holds for all other "transition probabilities".

If we associate A with S_2, B with S_1, and C with S_0 we have the following. If the minimal number of zeros a member of the current population contains, equals i, then the population satisfies a condition that is associated with S_i. We choose values for p_{S_2,S_1}, p_{S_1,S_0}, and p_{S_1,S_2} in the following way. Let P_A denote the set of all populations satisfying condition A. Let P_B and P_C denote the according sets for conditions B and C. We denote the change from a population P_1 to another population P_2 in one generation by $P_1 \to P_2$. We want p_{S_2,S_1} to be a lower bound on the probability to come from a population satisfying condition A to a population satisfying condition B, i. e. $p_{S_2,S_1} \leq \min\left\{\sum_{P_2 \in P_B} \text{Prob}\left(P_1 \to P_2\right) \mid P_1 \in P_A\right\}$. We want p_{S_1,S_0} to be a lower bound on the probability to come from a population satisfying condition B to a population satisfying condition C, thus we want to have $p_{S_1,S_0} \leq \min\left\{\sum_{P_2 \in P_C} \text{Prob}\left(P_1 \to P_2\right) \mid P_1 \in P_B\right\}$. Finally, we want p_{S_1,S_2} to be an upper bound on the probability to come from a population satisfying condition B to a population satisfying condition A, that is $p_{S_1,S_2} \geq \max\left\{\sum_{P_2 \in P_A} \text{Prob}\left(P_1 \to P_2\right) \mid P_1 \in P_B\right\}$.

Such lower bounds for p_{S_2,S_1} and p_{S_1,S_0} are easy to find. Introducing a string with exactly $n-1$ ones into the population can be achieved in the following way. First, one selects the current best member of the population, which is a string with exactly $n-2$ ones. As in the first phase, either the the whole population consists of strings with $n-2$ ones and have current best function value. In this case we obviously select such a string with probability 1. Otherwise, we apply Lemma 6 b) and have $f_1 = n^{2(n-2)}$ and $f_2 \leq n^{2(n-3)}$. Thus, the probability for this event is bounded below by $1 - N/n^2$ in any case. Then exactly one of the two bits with value zero is mutated. This event has probability $(2/n)(1-1/n)^{n-1}$. Finally, this child is not deleted from the (at this moment enlarged) population. We apply Lemma 7 a) and have $1 - 2/N$ as a lower bound. Therefore, we have

$$\left(1 - \frac{1}{n^{3/2}}\right) \cdot \frac{1}{2en} \cdot \left(1 - \frac{2}{\sqrt{n}}\right) = \Omega\left(\frac{1}{n}\right)$$

as a lower bound for p_{S_2,S_1}.

For p_{S_1,S_0} we consider the case that a string with $n-1$ ones is selected as the parent and the only bit with value 0 is mutated, only. The probability to select such a string is bounded below by

$$\frac{n^{2(n-2)} - 1}{N n^{2(n-2)}} = \frac{1}{\sqrt{n}} - \frac{1}{n^{2(n-2)+1/2}} \geq \frac{1}{2\sqrt{n}}$$

due to Lemma 6 a). Thus, we have $\left(1/\left(2\sqrt{n}\right)\right) \cdot (1/n) \cdot (1-1/n)^{n-1} = \Omega\left(1/n^{3/2}\right)$ as a lower bound on p_{S_1,S_0}.

Finally, we need an upper bound for p_{S_1,S_2}. Obviously, a necessary condition for the event we consider is that the only string with $n-1$ ones that is in the population is selected for deletion. We distinguish two different cases. First, assume that at least one of the strings in the set $\{x_2, x_3, \ldots, x_{N+1}\}$ (in line 7 of Algorithm 2) contains less than $n-2$ ones. This can be due to the fact that such a string was already member of the population or due to the fact that such a child was created. In this case the probability to select the string with $n-1$ ones for deletion is bounded above by $1/n^2$ due to Lemma 7 b). Thus, in the case that a string with less than $n-2$ ones is created by mutation we have $1/n^2$ as an upper bound on p_{S_1,S_2}.

Now, we consider the second case, where except for the one string x with $n-1$ ones all other members of the population contain exactly $n-2$ ones. First of all, the probability to select the string x for deletion is bounded above by $2/N$ due to Lemma 7 a). We consider two sub-cases with respect to the parent of the newly created child. If the parent is a string with $n-2$ ones, then the probability to create a child different from the parent with $n-2$ ones is bounded above by $2/n$, since at least one of the two bits with value zero has to flip. Otherwise, either a child with a number of ones that is different from $n-2$ is created or we have the case of a replication that is not allowed due to the definition of Algorithm 2. If, on the other hand, the parent is the only string with $n-1$ ones, it is obviously a necessary condition that this string is selected for reproduction. The probability of such a selection is bounded above by $\left(n^{2(n-2)} - 1\right) / \left(N n^{2(n-2)}\right) \leq 1/\sqrt{n}$ due to Lemma 6 a). Therefore, we have $\max\left\{1/n^2, (2/\sqrt{n}) \cdot 2/n, (2/\sqrt{n}) \cdot 1/\sqrt{n}\right\} = 2/n$ as an upper bound on p_{S_1,S_2} in any case.

We combine what we have and get $\text{E}\left(T_{S_2}\right) = \text{O}\left(n^{3/2}\right)$ leading to $\text{E}\left(T_{S_1}\right) = \text{O}\left(n\right) + \text{O}\left(n^{3/2}\right) = \text{O}\left(n^{3/2}\right)$ which is also an upper bound on the expected length of the second phase. Together with the upper bound on the expected length of the first phase we obtain the desired result. □

We see, that the EA optimizes SJump_{2,n^2} on average $\Omega\left(\sqrt{n}\right)$-times faster than the (1+1) EA. Note, that this result depends on the appropriate choice of the population size N. For $N = \Theta\left(\sqrt{n}\right)$ similar results can be proven. Note, however, that already for $N \geq n$ we cannot prove any better bound then $\Omega\left(n^2\right)$.

Thus, Theorem 5 is a result with two problems. First, the performance of an EA should be less dependent on the choice of the parameterization. Second, since EAs are heuristics and we do not expect optimal per-

formance, a gap of size $\sqrt{n}$ is not very convincing. Fortunately, the ideas and methods we developed so far, are sufficient to deliver much stronger results.

We consider functions $\text{SJUMP}_{k,s}\colon \{0,1\}^n \to \mathbb{R}$ with $k > 1$, where k is not too large, and $s \geq n^2$. It will turn out that $k = \text{O}\left((\log n) / \log\log n\right)$ is an appropriate choice. Similar to the proof of Theorem 5 we consider a special Markov chain and describe a tight connection between the expected run time of the EA and the expected absorption time of the Markov chain. Since the idea concerning the advantage of a population is the same, the number of states of the Markov chain equals $k+1$, just as in the proof of Theorem 5 (see Figure 1).

We find appropriate lower bounds on $p_{S_i,S_{i-1}}$ for all $i \in \{1,2,\ldots,k\}$ and appropriate upper bounds on p_{S_i,S_k} for all $i \in \{1,2,\ldots,k-1\}$ similar to the proof of Theorem 5. It is quite easy to see and straightforward to show, that

$$\left(\prod_{0<i<k} \frac{p_{S_i,S_{i-1}}}{p_{S_i,S_{i-1}} + p_{S_i,S_k}}\right)^{-1} \cdot \left(\sum_{0<i\leq k} \frac{1}{p_{S_i,S_{i-1}}}\right)$$

is an upper bound on the expected absorption time. Combining all these findings yields the following theorem. A rigorous proof is contained in a technical report.

Theorem 8. *Consider Algorithm 5 on the function* $\text{SJUMP}_{k,s}$ *with* $k > 1$, $k = O\left((\log n) / \log\log n\right)$, *and* $s \in \mathbb{N} \setminus \{1\}$ *with* $s \geq n^2$. *The expected run time of this EA with population size* N, *where* $n \leq N \leq \sqrt{s}$ *holds, is* $O\left(nN(c\cdot k)^{k+1}\right) = n^{O(1)}$, *where* c *is some positive constant.*

Let $k = \Theta\left((\log n)/\log\log n\right)$ and $s = \Omega\left(n^{\log n}\right)$. Then we have the following result. Using any polynomial population size $N \geq n$, the population-based evolutionary algorithm has polynomial expected run time on $\text{SJUMP}_{s,k}$, whereas the (1+1) EA has expected run time $\Theta\left(n^{(\log n)/\log\log n}\right)$, which is super-polynomial. Thus, there is a whole family of objective functions, where the expected run time can be tremendously reduced by employing a population instead of just one single individual.

4 CONCLUSIONS

We investigated the question whether the use of a population can by itself be advantageous even without using crossover. We measured the computational cost by the number of function evaluations. We considered the maximization of pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$ and proved for one example that the appropriate use of a population can speed-up optimization by a factor of $\Omega\left(\sqrt{n}\right)$. We could even strengthen this result, by proving, that the use of a large enough population of polynomial size can reduce the expected run time from super-polynomial to polynomial. This is the first such result that has been rigorously proven.

It remains open whether functions can be found where the advantage due to the use of a population (without employing crossover) is even exponential. It would be interesting to find a lower bound on the population size for the case where the use of a population is of substantial benefit. Moreover, we would like to identify other circumstances under which populations are advantageous. Note, that in order to do a fair comparison one has to allow the (1+1) EA to make use of restart mechanisms.

Acknowledgments

This work is based upon an idea of Jon Rowe (2000). He described a first intuition under what circumstances a carefully chosen EA may have an advantage compared to a random mutation hill-climber This intuition is the basis of the paper.

The work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center "Computational Intelligence" (SFB 531).

References

D. Arnold and H.-G. Beyer (2000). Local performance of the $(\mu/\mu\text{I},\lambda)$-ES in a noisy environment. In W. Spears and D. Whitley (Eds.), *FOGA*. to appear.

T. Bäck, D. B. Fogel, and Z. Michalewicz (Eds.) (1997). *Handbook of Evolutionary Computation*. Institute of Physics.

H.-G. Beyer (1993). Some asymptotical results from the $(1,+\lambda)$-theory. *Evolutionary Computation 1*(2), 165–188.

H.-G. Beyer (1995a). Toward a theory of evolution strategies: On the benefit of sex — the $(\mu/\mu,\lambda)$-theory. *Evolutionary Computation 3*(1), 81–111.

H.-G. Beyer (1995b). Toward a theory of evolution strategies: The (μ,λ)-theory. *Evolutionary Computation 2*(4), 381–407.

S. Droste, T. Jansen, and I. Wegener (1998). A rigorous complexity analysis of the (1+1) evolutionary algorithm for linear functions with Boolean inputs.

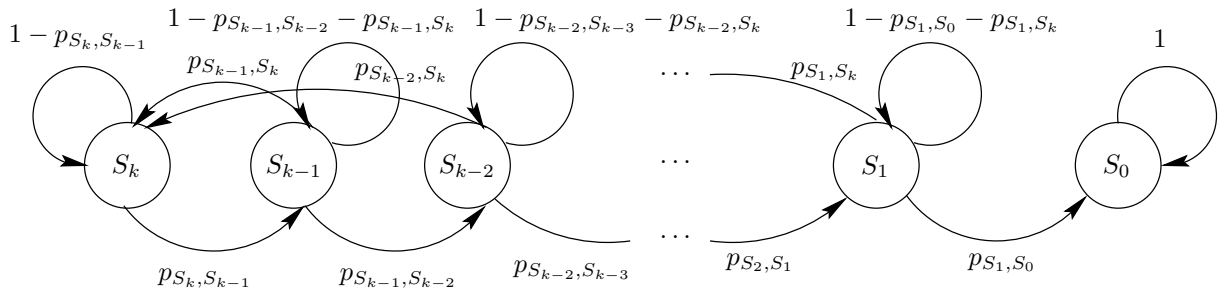


Figure 1: Markov chain modeling Algorithm 2 on $\text{SJUMP}_{k,s}$.

In D. B. Fogel, H.-P. Schwefel, T. Bäck, and X. Yao (Eds.), *Proceedings of the IEEE International Conference on Evolutionary Computation* (*ICEC '98*), 499–504. IEEE Press.

J. Garnier, L. Kallel, and M. Schoenauer (1999). Rigorous hitting times for binary mutations. *Evolutionary Computation 7*(2), 173–203.

T. Jansen and I. Wegener (1999). On the analysis of evolutionary algorithms — a proof that crossover really can help. In J. Nešetřil (Ed.), *Proceedings of the 7th Annual European Symposium on Algorithms* (*ESA '99*), Volume 1643 of *Lecture Notes in Computer Science*, 184–193. Springer, Berlin.

A. Juels and M. Wattenberg (1995). Hillclimbing as a baseline method for the evaluation of stochastic optimization algorithms. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 8*, 430–436. MIT Press, Cambridge, Mass.

M. Mitchell (1995). *An Introduction to Genetic Algorithms*. MIT Press.

M. Mitchell, S. Forrest, and J. H. Holland (1992). The royal road function for genetic algorithms: Fitness landscapes and GA performace. In F. J. Varela and P. Bourgine (Eds.), *Proceedings of the First European Conference on Artificial Life*, 245–254. MIT Press, Cambridge, MA.

M. Mitchell, J. H. Holland, and S. Forrest (1994). When will a genetic algorithm outperform hill climbing? In J. Cowan, G. Tesauro, and J. Alspector (Eds.), *Advances in Neural Information Processing Systems*, ? Morgan Kaufman, San Francisco, CA.

R. Motwani and P. Rhagavan (1995). *Randomized Algorithms*. Cambridge University Press, Cambridge.

H. Mühlenbein (1992). How genetic algorithms really work. Mutation and hillclimbing. In R. Männer and R. Manderick (Eds.), *Proceedings of the 2nd Parallel Problem Solving from Nature* (*PPSN II*), 15–25. North-Holland, Amsterdam, Niederlande.

S. Ronald (1998). Duplicate genotypes in a genetic algorithm. In *Proceedings of the IEEE International Conference on Evolutionary Computation* (*ICEC '98*). IEEE Press, Piscataway, NJ.

J. Rowe (2000). Personal communication.

G. Rudolph (1997). How mutation and selection solve long path-problems in polynomial expected time. *Evolutionary Computation 4*(2), 195–205.

Dynamic Weighted Aggregation for Evolutionary Multi-Objective Optimization: Why Does It Work and How?

Yaochu Jin, Markus Olhofer and Bernhard Sendhoff
Future Technology Research
Honda R&D Europe (D) GmbH
63073 Offenbach/Main, Germany
Email: {yaochu.jin, markus.olhofer, bernhard.sendhoff}@hre-ftr.f.rd.honda.co.jp

Abstract

Evolutionary Dynamic Weighted Aggregation (EDWA) has shown to be both effective and computationally efficient [1] for multi-objective optimization (MOO). Besides, it was also found empirically and surprisingly that EDWA was able to deal with multi-objective optimization problems with a concave Pareto front, which has proved to be beyond the capability of the Conventional Weighted Aggregation (CWA) methods [2]. In this paper, a theory on why CWA fails for multi-objective problems with a concave Pareto front is provided schematically. According to this theory, it can easily be explained why EDWA has worked well for both convex and concave multi-objective problems. Simulation examples are conducted on various test functions to support our theory. It is concluded that EDWA is an effective and efficient method for solving multi-objective optimization problems.

1 Introduction

Evolutionary multi-objective optimization has been widely investigated in the recent years [3, 4]. Generally speaking, there are three main approaches to evolutionary multi-objective optimization, namely, weighted aggregation approaches, population-based non-Pareto approaches and Pareto-based approaches [5].

Conventional weighted aggregation (CWA) based approaches two main weaknesses. Firstly, aggregation based approaches can provide only one Pareto solution from one run of optimization. Secondly, it has been shown that weighted aggregation is unable to deal with multi-objective optimization problems with a concave Pareto front [2].

One effort using weighted aggregation based approach for multi-objective optimization (MOO) was reported in [6]. In that work, the weights of the different objectives are encoded in the chromosome to obtain more than one Pareto solution. Phenotypic fitness sharing is used to keep the diversity of the weight combinations and mating restrictions are required so that the algorithm can work properly.

An efficient and effective method called evolutionary dynamic weighted aggregation (EDWA) was proposed in [1]. The original idea in EDWA was straightforward, i.e. if the weights for the different objectives are changing during optimization, the optimizer will go through all points on the Pareto front. If the found non-dominated solutions are archived, the whole Pareto front can be achieved. This has been shown to be working well for both convex and concave Pareto fronts.

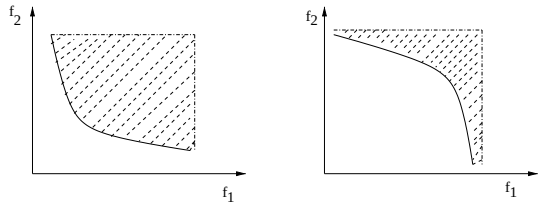
In this paper, a theory on evolutionary multi-objective optimization using weighted aggregation is suggested. Based on this theory, the reason why EDWA is able to deal with MOO is revealed. Simulations are carried out on different test functions both to support our theory and to demonstrate the effectiveness of EDWA.

2 Multi-objective Optimization with Weighted Aggregation

2.1 Definition of Multi-objective Optimization

Consider a multi-objective optimization problem with k objectives $(f_i, i = 1, 2, ..., k)$ and n decision variables $(x_i, i = 1, 2, ..., n)$:

$$\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), ..., f_k(\mathbf{x})), \quad (1)$$



(a) Convex Pareto front (b) Concave Pareto front

Figure 1: Convex and concave Pareto fronts.

The target of the optimization is to minimize $f_i(\mathbf{x}), i = 1, 2, ..., k$ subject to

$$g_i(\mathbf{x}) \leq 0, i = 1, 2, ..., m. \quad (2)$$

Since the k objectives may be conflicting with each other, it is usually difficult to obtain the global minimum for each objective at the same time. Therefore, the target of MOO is to achieve a set of solutions that are Pareto optimal. The related concepts of Pareto dominance, Pareto optimality, Pareto optimal set and Pareto front are defined as follows [4]:

Pareto dominance: A vector $\mathbf{u} = (u_1, ..., u_k)$ is said to dominate $\mathbf{v} = (v_1, ..., v_k)$ if and only if $u_i \leq v_i, i = 1, 2, ..., k$ and there exists at least one element with $u_i < v_i$.

Pareto optimality: A solution $\mathbf{x}$ is said to be Pareto optimal if and only if there does not exist another solution $\mathbf{x}'$ so that $\mathbf{f}(\mathbf{x})$ is dominated by $\mathbf{f}(\mathbf{x}')$. All the solutions that are Pareto optimal for a given multi-objective optimization problem are called the Pareto optimal set ($\mathcal{P}^\star$).

Pareto front: For a given multi-objective optimization problem and its Pareto optimal set $\mathcal{P}^\star$, the Pareto front ($\mathcal{PF}^\star$) is defined as:

$$\mathcal{PF}^\star = \{\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), ..., f_k(\mathbf{x})) | \mathbf{x} \in \mathcal{P}^\star\}. \quad (3)$$

There are generally convex and concave Pareto fronts. A Pareto front ($\mathcal{PF}^\star$) is said to be convex if and only if $\forall \mathbf{u}, \mathbf{v} \in \mathcal{PF}^\star, \forall \lambda \in (0, 1), \exists \mathbf{w} \in \mathcal{PF}^\star : \lambda||\mathbf{u}|| + (1 - \lambda)||\mathbf{v}|| \geq ||\mathbf{w}||$.

On the contrary, a Pareto front is said to be concave if and only if $\forall \mathbf{u}, \mathbf{v} \in \mathcal{PF}^\star, \forall \lambda \in (0, 1), \exists \mathbf{w} \in \mathcal{PF}^\star : \lambda||\mathbf{u}|| + (1 - \lambda)||\mathbf{v}|| \leq ||\mathbf{w}||$.

For example, Fig.1(a) is a convex Pareto front and Fig.1(b) is a concave Pareto front. Of course, a Pareto front can be partially convex and partially concave.

2.2 Conventional Weighted Aggregation for MOO

Conventional Weighted Aggregation (CWA) is a straightforward approach to multi-objective optimization. In this method, the different objectives are summed up to a single scalar with a prescribed weight

$$F = \sum_{i=1}^{k} w_i f_i(\mathbf{x}), \quad (4)$$

where w_i is the non-negative weight for objective $f_i(\mathbf{x}), i = 1, ..., k$. Usually, *a priori* knowledge is needed to specify the weights. During the optimization, the weights are fixed in conventional weighted aggregation method.

Using this method, only one Pareto optimal solution can be obtained with one run of the optimization algorithm. In other words, if one intends to obtain different Pareto solutions, one has to run the optimizer several times. This is of course not allowed in a lot of real world problems because it usually takes too much time to run the optimization more than once.

What is worse, efficiency is not the only problem for CWA. It was pointed out that CWA is not able to obtain the Pareto solutions that are located in the concave region of the Pareto front [2].

However, it is not as straightforward as one might imagine to explain the reason why the solutions in the concave region of the Pareto front cannot be obtained using CWA. One attempt to explain this problem is illustrated in Fig. 2, which was provided in [2]. In the figure, line L denotes solutions with the same cost and the slope of the line is determined by the weights. According to this theory, the solutions in the concave region between point A and B cannot be reached by CWA based methods. Unfortunately, this illustration is incorrect because the solutions outside the shaded area are unreachable anyway and therefore, it is impossible for the optimizer to proceed towards the Pareto front from the origin, in particular for minimization problems.

In [5], another illustration as shown in Fig. 3 is used. However, from this illustration, it is still unclear why the solutions in the concave region are not obtainable with CWA methods. Further explanations are provided as follows. In Fig. 4 (a), it can be seen that the line with equal cost will converge to a point on a convex Pareto front when the slant of the line is given, that is, when the weights are fixed. In contrast, the line will continue to move after it reaches a point (point C, which is corresponding to the given weights)

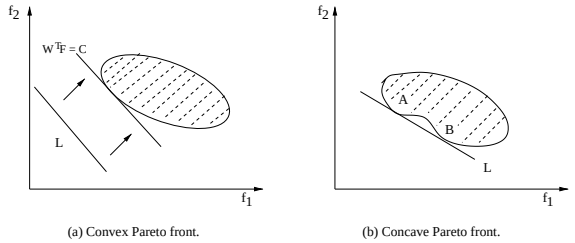


Figure 2: Geometrical representation of weighted sum approach abstracted from [2].

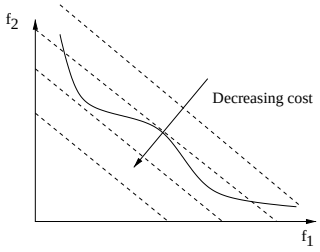


Figure 3: Line of equal cost introduced by the weighted-sum approach abstracted from [4].

in the concave region of the Pareto front, until no further minimization of the cost is possible. Finally, the obtained solution will be either A or B.

In the following, a new explanation for this problem is suggested. In our opinion, whether CWA is able to converge to a Pareto-optimal solution depends on the stability of the Pareto solution corresponding to the given weight combination. If the Pareto solution corresponding to a given weight combination is a stable minimum, then it can be obtained with CWA. To explain this further, let us have a look at the problem from another point of view. We first discuss a convex Pareto front. For a two-objective problem, if the Pareto front is presented in the conventional way, as shown in Fig. 5 (a), then point B is the stable minimum on the Pareto front. That is to say, point B will be the solution obtained by CWA given the weight combination of $(w_1, w_2) = (0, 1)$. If the weight combination is changed in optimization, it is equivalent to rotating the coordinate system together with the Pareto front. Thus, when w_1 decreases and w_2 increases, it is equal to rotate the coordinate system counter-clockwise. If $f_1^\star = f_2^\star$, then for a given weight combination of $(w_1, w_2) = (0.5, 0.5)$, the coordinate system rotates 45 degrees. In this case, C is the stable minimum of the Pareto front that will be obtained using CWA with $(w_1, w_2) = (0.5, 0.5)$, as shown in Fig. 5 (b). Obviously, for a weight combination of $(w_1, w_2) = (1, 0)$, A is the stable minimum and the coordinate system rotates 90 degrees. Therefore, different Pareto solutions will be obtained using the conventional weight aggregation with different weight combinations if the Pareto front is convex. Since the weights are always non-negative, the maximal rotation angle is 90 degree. Without considering the time consumption, the whole Pareto front can be obtained by running the optimizer as many times as possible.

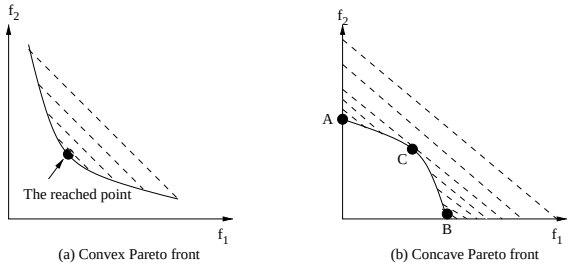


Figure 4: Conventional weighted aggregation for MOO. (a) Convex Pareto front, (b) Concave Pareto front.

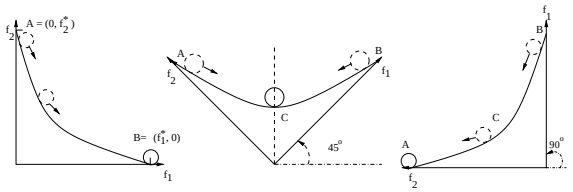


Figure 5: Convex Pareto front. All Pareto solutions are stable minimum when the coordinate system rotates. (a) 0 degree; (b) 45 degree; (c) 90 degree.

Now let us have a look at a concave Pareto front. As illustrated in Fig. 6, all solutions located in the concave region of the Pareto front are unstable when the weight combination changes. As explained above, Fig. 6 (a) corresponds to a weight combination of $(w_1, w_2) = (0, 1)$ and the solution will be point B. For all weight combinations that corresponds to a rotation angle between 0 and 45 degrees, the solution to be obtained will be B, whereas for all weight combinations that correspond to a rotation angle between 45 and 90 degrees, the solution to be obtained will be A. The weight combination that corresponds to a rotation angle of 45 degree (if $f_1^\star = f_2^\star$) is a dividing point (Point C). If the weight combination exactly corresponds to this dividing point, the result of the optimization can either be A or B, depending on the initial condition and dynamics of the optimizer. As a conclusion, only point A and B are stable minima on the Pareto front no matter how the weight combination changes.

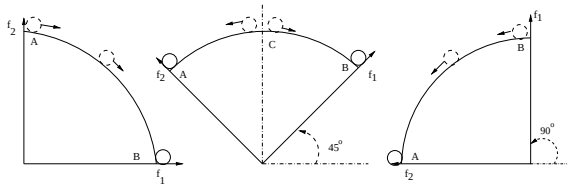


Figure 6: Concave Pareto front. All Pareto solutions are unstable minimum except the two points on both ends when the coordinate system rotates. (a) 0 degree; (b) 45 degree; (c) 90 degree.

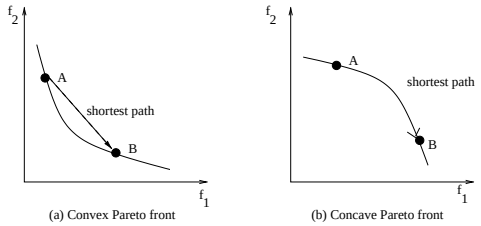


Figure 7: For a convex Pareto front (a), it is not the shortest path between two points, whereas for a concave Pareto front (b), it is the shortest.

According to the above discussions, we can draw the following conclusions:

- For a convex Pareto front, each weight combination corresponds to a stable minimum on the Pareto front.
- For a concave Pareto front, all solutions with exception to the two points on the two ends are unstable when the conventional weighted aggregation is used. Therefore, an optimizer is unable to converge to the Pareto solution corresponding to the weight combination.
- When the Pareto front is rotated slowly from 0 degree to 90 degree, the optimizer will go along the Pareto front from one stable minimum to another, *once it reaches any point of the Pareto front.* If the Pareto front is convex, the moving speed is determined by the change of weights. If the Pareto font is concave, the optimizer will stay on one stable minimum until this point becomes unstable. In this case, the optimizer will move *along the Pareto front* to the next stable minimum.

3 Evolutionary Dynamic Weighted Aggregation

As it is pointed out in the last section, if we rotate the Pareto front 90 degree, the optimizer will go from one stable optimum to another. This can be done in two ways:

- After the optimizer has converged to one stable minimum, the Pareto front is rotated 90 degrees abruptly. In the two-objective case, this corresponds to the situation where w_1 is changed from 0 to 1 and w_2 from 1 to 0. We call it Bang-bang Weighted Aggregation (BWA).
- The Pareto front is rotated gently, that is, the weights are changed gradually. In this case, the optimizer will traverse the whole Pareto front and all the solutions on the front will be obtained. This was denoted as Generation-based Periodical Variation of the Weights in [1]. In this paper, it is called Dynamic Weighted Aggregation (DWA).

In both cases, the weights are changed periodically. This may be helpful if the Pareto front is not uniform. By uniform, we mean that if the distance in the weight space is the same, then the distance on the Pareto front is also the same.

3.1 Bang-bang Weighted Aggregation

Bang-bang weighted aggregation (BWA) can be seen as a test of our theory proposed in the last section. According to our theory, it is also *possible* to obtain the whole Pareto front when we rotate it 90 degrees abruptly, no matter whether it is convex or concave. However, we expect that the optimizer may not necessarily keep moving along the Pareto front if it is convex, because the Pareto front is not the shortest feasible path from one stable point to another, refer to Fig. 7 (a). Very interestingly, if the Pareto front is concave, the optimizer should keep moving along the Pareto front because it provides the shortest feasible path from one stable point to another, as illustrated in Fig. 7 (b).

A bang-bang change of weights can be realized in the following way for a two-objective minimization problem:

$$w_1(t) = \text{sign}(\sin(2\pi t/F)) \quad (5)$$

$$w_2(t) = 1.0 - w_1, \quad (6)$$

where t is the generation index and F is the frequency of the weight change. It is clear that F should be large enough to allow the optimizer to move from one stable point to another.

3.2 Dynamic Weighted Aggregation

In dynamic weighted aggregation (DWA), the weights are changed gradually. This slow change of the weights will force the optimizer to keep moving on the Pareto front if it is convex. If it is concave, the performance of DWA may not have much difference from that of the BWA. This can be realized as follows:

$$w_1(t) = |\sin(2\pi t/F)|, \quad (7)$$
$$w_2(t) = 1.0 - w_1(t), \quad (8)$$

where t is the number of generation. It is noticed that $w_1(t)$ changes from 0 to 1 periodically. The change frequency can be adjusted by F. The frequency should not be too high so that the algorithm is able to converge to a minimum. On the other hand, it seems reasonable to let the weight change from 0 to 1 twice during the whole optimization. In the simulation described in Section 5.3 and 5.4, F is set to 100 for BWA and 200 for DWA, so that for both methods, the Pareto front rotates three times in 150 generations.

3.3 An Archive of Pareto Solutions

In both BWA and DWA, the population is not able to keep all the found Pareto solutions, although it is able to traverse the Pareto front dynamically. Therefore, it is necessary to record the Pareto solutions that have been found so far. To this end, it is necessary to maintain an archive of the Pareto-optimal solutions. The pseudo-code for building the archive is listed in Algorithm 1. The similarity is measured by the Euclidean distance in the fitness space.

Algorithm 1 Pseudo-code for maintaining an archive of Pareto solutions.

```
for each individual o in offspring population do
  if (o dominates an individual in parent population p) and (o is not dominated by any solutions in the archive) and (o is not similar to any solutions in the archive) then
    if archive is not full then
      add o to the archive
    else if o dominates any solution a in the archive then
      replace a with o
    else if any solution a1 in the archive dominates another solution a2 then
      replace a2 with o
    else
      discard o
    end if
  else
    discard o
  end if
end for
for each solution in the archive do
  if solution a1 dominates a2 then
    remove a2
  end if
end for
```

4 Test Functions

To evaluate our theory and to demonstrate the effectiveness of our methods, simulations are carried out on five test functions. The first three test functions are taken from [7, 8] and the fourth test function is adapted from test functions F_2 and F_3 so that its Pareto front is partially convex and partially concave. F_5 has a discontinuous Pareto front, which is used to test how the methods behave when the Pareto front is discontinuous. Note that for all test functions, $x_i \in [0, 1]$.

- The first test function (F_1) used here is the second function in [8] and we extend it to an n-dimensional. The Pareto front of this function is uniform.

$$f_1 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 \quad (9)$$
$$f_2 = \frac{1}{n}\sum_{i=1}^{n} (x_i - 2.0)^2 \quad (10)$$

- The second test function (F_2) is the first function in [7], which has a convex but non-uniform Pareto front:

$$f_1 = x_1 \quad (11)$$
$$g(x_2, ..., x_n) = 1.0 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \quad (12)$$
$$f_2 = g \times (1.0 - \sqrt{f_1/g}). \quad (13)$$

- The third test function (F_3) is the second function in [7], which has a concave Pareto front:

$$f_1 = x_1 \quad (14)$$
$$g(x_2, ..., x_n) = 1.0 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \quad (15)$$
$$f_2 = g \times (1.0 - (f_1/g)^2). \quad (16)$$

- The fourth test function (F_4) adapted from F_2 and F_3, which has a Pareto front that is neither

purely convex nor purely concave:

$$f_1 = x_1 \quad (17)$$

$$g(x_2, ..., x_n) = 1.0 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \quad (18)$$

$$f_2 = g \times (1.0 - \sqrt[4]{f_1/g} - (f_1/g)^4). \quad (19)$$

- The fifth test function (F_5) is the third function in [7], whose Pareto front consists of a number of separated convex parts:

$$f_1 = x_1 \quad (20)$$

$$g(x_2, ..., x_n) = 1.0 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \quad (21)$$

$$f_2 = g \times (1.0 - \sqrt{f_1/g} - (f_1/g) \sin(10\pi f_1)). \quad (22)$$

5 Simulation Studies

5.1 The Evolution Strategies

In the standard evolution strategy, the mutation of the objective parameters is carried out by adding an $N(0, \sigma_i^2)$ distributed random number. The step size σ_i is also encoded in the genotype and subject to mutations. A standard evolution strategy can be described as follows:

$$\sigma_i(t) = \sigma_i(t-1)\exp(\tau' z)\exp(\tau z_i) \quad (23)$$

$$\mathbf{x}(t) = \mathbf{x}(t-1) + \tilde{\mathbf{z}} \quad (24)$$

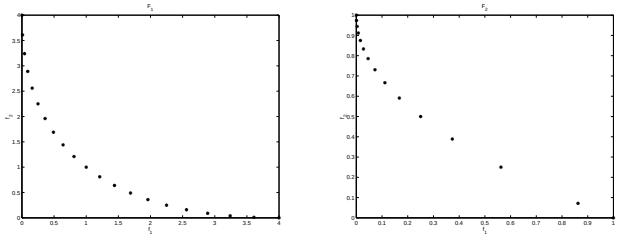
where $\mathbf{x}$ is an n-dimensional parameter vector to be optimized, $\tilde{\mathbf{z}}$ is an n-dimensional random number vector with $\tilde{\mathbf{z}} \sim N(\mathbf{0}, \boldsymbol{\sigma}(t)^2)$, z and z_i are normally distributed random numbers with $z, z_i \sim N(0,1)$. Parameters τ, τ' and σ_i are the strategy parameters, where σ_i is mutated as in equation (24) and τ, τ' are constants as follows:

$$\tau = \left(\sqrt{2\sqrt{n}}\right)^{-1}; \ \tau' = \left(\sqrt{2n}\right)^{-1} \quad (25)$$

There are several extensions to the above standard ES. In this work, the standard (μ, λ)-ES [9] is employed.

5.2 Conventional Weighted Aggregation

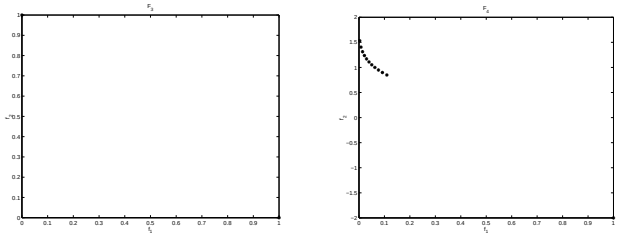
We first employ CWA to obtain the Pareto front. As we mentioned above, we have to run the optimizer more than once if we attempt to obtain more than one solution. In this work, the algorithm is run for 20 times for test functions F_1, F_2 and F_3, and 40 times for test function F_4. Since the Pareto solutions are not uniformly distributed on the Pareto front corresponding to a uniformly distributed weight combinations, smaller weight change is needed to obtain the solutions in the convex region of the Pareto front in function F_4. In all the simulations, the dimension n is set to 2.



(a) (b)

Figure 8: CWA for F_1 and F_2. The results are collected from 20 runs of the optimization.

The results on F_1 and F_2 are given in Fig. 8. Since both functions are convex, the CWA based approach is able to obtain different Pareto solutions with different weights. We see that the distribution of the solutions from F_2 is not uniform, although the distribution of the weights is uniform.



(a) (b)

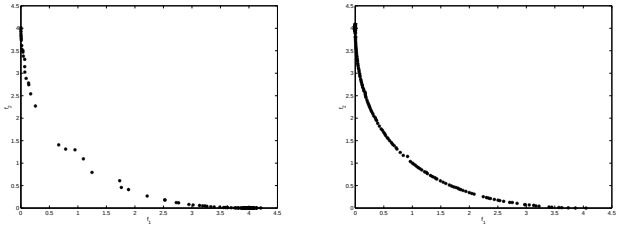
Figure 9: CWA for F_3 and F_4. 20 runs are carried out for F_3 and 40 runs are carried for F_4.

Fig. 9 provides the results on F_3 and F_4. Since the Pareto front of F_3 is concave, we can only obtain two solutions, whereas for F_4, the solutions in the convex region are obtained and those in the concave region are not obtained, which is expected from our discussion in Section 3.

5.3 Bang-bang Weighted Aggregation and Dynamic Weighted Aggregation

In this part, we intend to empirically support our theory on multi-objective optimization by showing that bang-bang weighted aggregation is able to obtain the Pareto set, in particular for concave Pareto fronts. At

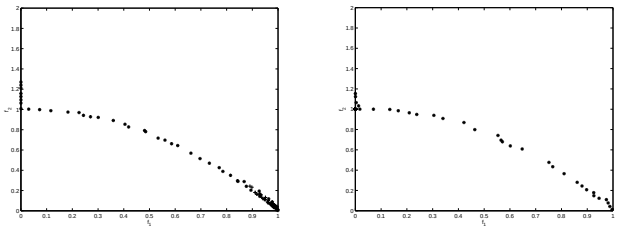
the same time, the performance of both BWA and DWA are compared for different situations. For both methods, 150 generations are run so that the weight can switch three times during optimization, as mentioned in Section 3.2.



(a) (b)

Figure 10: Results on F_1: (a) BWA and (b) DWA.

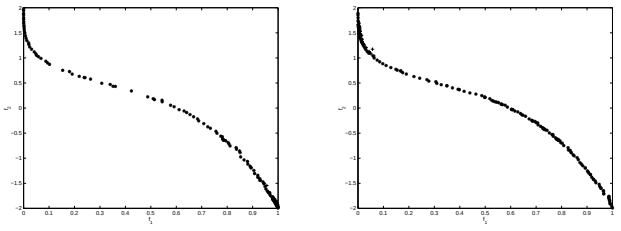
According to our theory, BWA may perform worse than DWA for convex Pareto fronts, because the Pareto front between two stable points is not the shortest feasible path. This can be seen from the results on F_1, which are shown in Fig. 10. However, when the Pareto front is concave, the performances of BWA and DWA are similar, as shown in Fig. 11. This is consistent with our theory.



(a) (b)

Figure 11: Results on F_3: (a) BWA and (b) DWA.

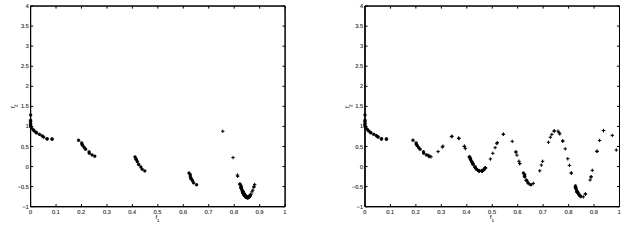
Test function F_4 has a partially convex and partially concave Pareto front. Since its convex part is relatively small, there is no essential discrepancy between the results from BWA and DWA, see Fig. 12.



(a) (b)

Figure 12: Results on F_4: (a) BWA and (b) DWA.

All the above test functions, F_1, F_3 and F_4 have a continuous Pareto front. It is desirable to investigate how our methods work for discontinuous Pareto fronts, particularly when BWA is used. In Fig. 13 (a), we see that BWA has successfully obtained the discontinuous Pareto front. Amazingly, the algorithm is able to move from one part of the Pareto front to another through a *bridge* that connects the different parts of the Pareto front, which is shown in Fig. 13 (b) caught by a snapshot during optimization.



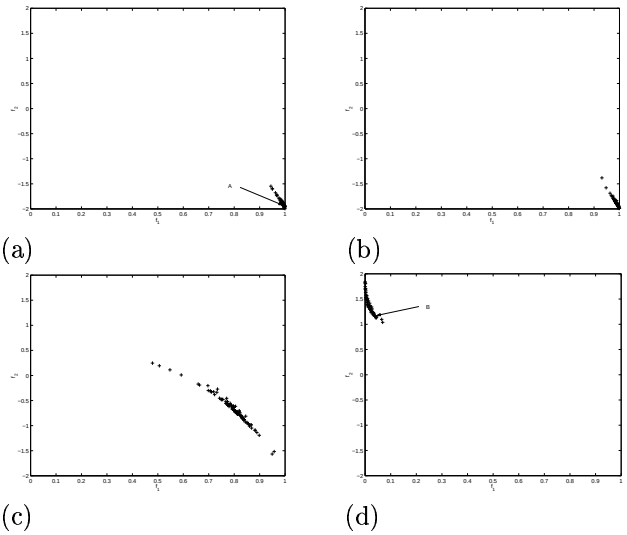
(a) (b)

Figure 13: BWA for a discontinuous Pareto front (F_5): (a) The obtained Pareto front; (b) A snapshot showing how the BWA moves between different parts of the Pareto front.

Similar results have been obtained on F_5 using DWA.

5.4 Discussions

From the simulation results, we can make the following observations:



(a) (b) (c) (d)

Figure 14: DWA for F_4: The optimizer starts to move along a concave Pareto front only when the rotation angle reaches a dividing point. (a) $w_1 = 0.0$; (b) $w_1 = 0.9$; (c) $w_1 = 0.92$ and (d) $w_1 = 1.0$.

- For convex Pareto fronts, DWA exhibits better performance than BWA. The reason is that the

optimizer will not necessarily keep moving along the Pareto front if BWA is used.

- For concave Pareto fronts, BWA and DWA show similar performance. However, BWA may be more efficient than DWA when the Pareto front is concave. This is due to the fact that when the Pareto front rotates, the optimizer stays at one stable solution until a rotation angle corresponding to the dividing point has been reached, as discussed in Section 2.2. Let us have a look at the results from applying the DWA to F_4. In generation 100, $w_1 = 0.0$ and the optimizer is around the stable point A, see Fig. 14 (a). When the evolution proceeds, the optimizer remains on point A until in generation 135, when $w_1 = 0.90$, see Fig. 14 (b). In generation 137, the optimizer has moved through the most part of the concave region (Fig. 14 (c)), where $w_1 = 0.92$. Finally, in generation 150, the optimizer is around the other stable point of the concave region, i.e., point B in Fig. 14 (d). It should be noticed that the solutions in the archive are not shown in Fig. 14.

In application, if one does not know in advance if the Pareto front is convex or concave, DWA is recommended to ensure that the optimizer will move along the Pareto front to obtain the whole set of Pareto solutions. However, if one knows that the Pareto front is concave, the BWA may need less time to achieve the whole Pareto set.

6 Conclusion

Multi-objective optimization using weighted aggregation based approaches is revisited. The problem of concave Pareto fronts in MOO is discussed and a theory why CWA based approaches are unable to obtain the solutions in the concave region of the Pareto front is proposed. An evolutionary dynamic weighted aggregation (EDWA) is proposed to obtain Pareto solutions in one run, no matter whether they are located in the convex or concave region of the Pareto front. The proposed method is shown to be not only efficient, meaning it is able to obtain Pareto solutions in one run of the optimization, but is also able to obtain the solutions located in the concave region of the Pareto front. This is very encouraging because the EDWA is computationally efficient and all existing evolutionary algorithms can be employed with minor modifications to change the weight dynamically during optimization.

However, theoretic work may be necessary to ascertain that when the weights changes and the optimizer moves from one stable minimum to another stable minimum, all the solutions in the concave region between the two minima are reached. The conclusion that local optima are concentrated in a very small region of the solution space [10] may be one support for the EDWA and vice versa, the successful of the EDWA is also a support for this conclusion.

ACKNOWLEDGMENTS

The authors would like to thank E. Körner and W. von Seelen for their support, T. Arima for his helpful comments and M. Hasenjäger and T. Okabe for stimulating discussions.

References

[1] Y. Jin, T. Okabe, and B. Sendhoff. Adapting weighted aggregation for multi-objective evolution strategies. In K. Deb, L. Thiele, and E. Zitzler, editors, *First International Conference on Evolutionary Multi-criterion Optimization*, Lecture Notes in Computer Science, pages 96–110, Zurich, Switzerland, 2001. Springer.

[2] P.J. Fleming. Computer aided control systems using a multi-objective optimization approach. In *Proc. IEE Control'85 Conference*, pages 174–179, Cambridge, U.K., 1985.

[3] C.A.C. Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):269–308, 1999.

[4] D. A. Van Veldhuizen and G. B. Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-art. *Evolutionary Computation*, 8(2):125–147, 2000.

[5] C. M. Fonseca and P. J. Fleming. Multiobjective optimization. In Th. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Evolutionary Computation*, volume 2, pages 25–37. Institute of Physics Publishing, Bristol, 2000.

[6] P. Hajela and C. Y. Lin. Genetic search strategies in multicriteria optimal design. *Structural Optimization*, 4:99–107, 1992.

[7] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolution algorithms: empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[8] J. D. Knowles and D. W. Corne. Approximating the nondominated front using the Pareto archived evolution strategies. *Evolutionary Computation*, 8(2):149–172, 2000.

[9] H.-P. Schwefel. *Evolution and Optimum Seeking.* Sixth-Generation Computer Technologies Series. John Wiley & Sons, Inc., 1994.

[10] P.C. Borges and M.P. Hansen. A basis for future successes in multiobjective combinatorial optimization. Technical Report IMM-REP-1998-8, Department of mathematical Modeling, Technical University of Denmark, 1998.

Main Vector Adaptation: A CMA Variant with Linear Time and Space Complexity

Jan Poland
University Tübingen, WSI RA
Sand 1, D - 72076 Tübingen
Germany
poland@informatik.uni-tuebingen.de

Andreas Zell
University Tübingen, WSI RA
Sand 1, D - 72076 Tübingen
Germany
zell@informatik.uni-tuebingen.de

Abstract

The covariance matrix adaptation (CMA) is one of the most powerful self adaptation mechanisms for Evolution Strategies. However, for increasing search space dimension N, the performance declines, since the CMA has space and time complexity $O(N^2)$. Adapting the *main mutation vector* instead of the covariance matrix yields an adaptation mechanism with space and time complexity $O(N)$. Thus, the main vector adaptation (MVA) is appropriate for large-scale problems in particular. Its performance ranges between standard ES and CMA and depends on the test function. If there is one preferred mutation direction, then MVA performes as well as CMA.

1 Introduction

Evolution Strategies need self adaptation, in order to apply to hard or badly scaled fitness functions. For a motivating example, consider the optimization of a prism lens from [4], chaper 9: Given is a glass block that consists of 19 prism segments. The thickness of the segments at both ends is variable. Hence, there are 20 object variables, which are tuned in order to focus the light rays and minimize the overall thickness of the lens (see Fig. 1, the exact fitness function is stated as f_{10} in Section 5). A simple ES with mutative or derandomized step length control finds the focus quite easily, but it fails to minimize the lens thickness. This is due to the fact that all object variables have to be reduced simultaneously in order to minimize the thickness, while any other mutation direction destroys the focus.

Other self adaptation algorithms fail in this situation,

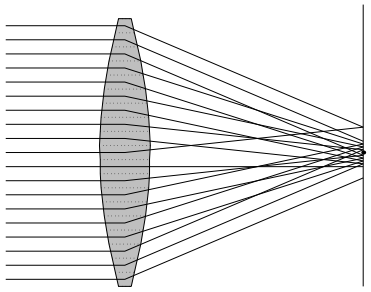


Figure 1: Optimization of a prism lens

too. One could expect for example an adaptation of the mutation mean (momentum adaptation) to be appropriate (compare e.g. [3]). However, our experiments with such an algorithm have not been successful. On the other hand, the covariance matrix adaptation (CMA), which adapts the covariance of the mutation, does work in this situation. In this paper, we will develop a new self adaptation algorithm that also works in this situation and is based on similar ideas as the CMA, but with less space and time consumption.

2 The CMA Algorithm

The covariance matrix adaptation (see [1] or [2]) is one of the most powerful self adaptation mechanisms today available for Evolution Strategies. While a simple ES uses a mutation distributen $N(0, \sigma^2 \cdot I)$ (where I is the identity matrix), the CMA-ES performes a $N(0, \sigma^2 \cdot C)$- distributed mutation, and the covariance matrix C is being adapted. This procedure is based on the following ideas (see [1]): Let $Z_1, \ldots, Z_n$ be independently $N(0,1)$ distributed, $z_1, \ldots, z_n \in \mathbb{R}^N$, $\sigma_1, \ldots, \sigma_n \in \mathbb{R}$ and

$$Z = \sum_{i=1}^{n} Z_i \cdot \sigma_i z_i \quad \text{and} \quad C = \sum_{i=1}^{n} \sigma_i^2 \cdot z_i z_i'.$$

Then Z is a normally distributed random vector with mean 0 and covariance C. On the other hand, any N-dimensional normal distribution $N(0, C)$ can be generated by such a sum by choosing for z_i the eigenvectors and for σ_i^2 the corresponding eigenvalues of C.

Thus, the offspring $\hat{x}$ can be created by adding a $N(0, \sigma^2 \cdot C)$ distributed random vector to the parent x, where $\sigma > 0$ is the global step size for x which can be adapted conventionally. If p_m denotes the mutation path, i.e. the (weighted) mean of the last successful steps, then C can be updated by

$$\hat{C} = (1 - c_{cov}) \cdot C + c_{cov} \cdot \hat{p}_m \hat{p}'_m,$$

where $c_{cov} > 0$ is a small constant and $\hat{C}$ is the covariance matrix for the next generation. The path p_m is updated by a similar formula:

$$\hat{p}_m = (1 - c_m) \cdot p_m + c_m^u \cdot \sigma^{-1}(\hat{x} - x)$$

(note that $\sigma^{-1}(\hat{x} - x)$ is $N(0, C)$ distributed). Again, $c_m > 0$ is a small constant, while $c_m^u = \sqrt{c_m(2 - c_m)}$, which assures that p_m and $\hat{p}_m$ are identically distributed if p_m and $\sigma^{-1}(\hat{x} - x)$ are independent and identically distributed. Hence, the path is not influenced by the global step size σ.

This covariance matrix adaptation procedure has turned out to be very efficient and is successful in cases where the standard ES breaks down. In particular, the CMA makes the strategy invariant against any linear transformation of the search space. Moreover, the covariance matrix approximates the inverse Hessian matrix for functions with sufficient regularity properties. Thus, the CMA can be considered as an evolutionary analogon to quasi Newton optimization algorithms.

The main drawback of the CMA comes with increasing dimension N of the search space. The storage space and the update time for the covariance matrix have complexity $O(N^2)$, while the computation of the eigenvectors and eigenvalues is even $O(N^3)$. This can be reduced to $O(N^2)$ by executing the step for example after $N/10$ generations instead of every generation, which does no severe damage. In any case, for large N, the CMA performance declines rapidly, compare also Fig. 7. There are other self adaptation mechanisms which are similar to CMA, such as the rotation angle adaptation (see [5]). This algorithm has quadratic space and time complexity as well and shows a poorer performance in general.

3 Main Vector Adaptation

For many functions the advantage of the CMA compared to a conventional ES is given by the fact that the CMA finds the *preferred mutation direction*, while all other directions are not acceptable. An instance is the lens optimization (see the introduction). In these cases, it should be sufficient to adapt one vector instead of an entire matrix in order to find this direction. This is done basically by the simple formula $\hat{v} = (1 - c_v) \cdot v + c_v \cdot \hat{p}_m$, where p_m is the path as before and $c_v > 0$ is a small constant. Then, the offspring $\hat{x}$ can be generated by $\hat{x} = x + \sigma \cdot Z + \sigma \cdot Z_1 \cdot v$, where $Z \sim N(0, I)$ and $Z_1 \sim N(0, 1)$. We call v the *main (mutation) vector* and the algorithm *main vector adaptation* (MVA).

In order to make these formulas work in practice, we have to regard two details. First, the mutation is independent of the sign of the main vector v. However, in contrast to the update formula for the covariance matrix, the update of v depends on the sign of the path p_m. This can result in the annihilation of subsequent mutation steps and inhibits the adaptation of v, in particular for difficult functions such as the sharp ridge f_7 (cf. Section 5). To avoid this breakdown, we simply flip v if necessary:

$$\hat{v} = (1 - c_v) \cdot \text{sign}(\langle v, p_m \rangle) \cdot v + c_v \cdot \hat{p}_m.$$

Here, $\langle \cdot, \cdot \rangle$ denotes the scalar product of two vectors.

The second problem to be fixed is the standard deviation of $Z + Z_1 \cdot v$ along the main vector v. Since $Z + Z_1 \cdot v \sim N(0, I + vv')$, its variance along v is $\sigma_v^2 = 1 + \|v\|^2$, hence $\sigma_v = \sqrt{1 + \|v\|^2}$. On the contrary, $\sigma_v = 1 + \|v\|$ would be desired, since this corresponds to the functioning of v as additional mutation in the main vector direction. Thus, we write

$$\hat{x} = x + \sigma \cdot (Z + Z_1 \cdot w_v \cdot v).$$

Letting $w_v = 1 + 2 \cdot \|v\|^{-1}$ yields $\sigma_v = 1 + \|v\|$, however, the experiments show that a constant $w_v = 3$ (corresponding to $\|v\| = 1$) yields the best results in general, occasionally, $w_v = 1$ is better.

Again, we point out that taking v (or anything similar) as the *mean vector* of the mutation does not yield an efficient algorithm! This is presumably due to the geometry of the high dimensional $\mathbb{R}^N$, where a nonzero mutation mean results in a shifted sphere, while the additional main vector mutation yields an ellipsoid as mutation shape.

4 The MVA-ES Algorithm

In order to adapt the mutation step size σ, we employ the same derandomized mechanism using a path p_σ, with the only difference that for p_σ the main vector v

is ignored. This is again a perfect analogy to the CMA ([1]). Thus, the complete mutation algorithm reads as follows.

Mutation.

1. Generate $Z \sim N(0, I)$
2. Generate $Z_1 \sim N(0, 1)$
3. $\hat{x} = x + \sigma \cdot (Z + Z_1 \cdot w_v \cdot v)$
4. $\hat{p}_\sigma = (1 - c_\sigma) \cdot p_\sigma + c_\sigma^u \cdot Z$
5. $\hat{\sigma} = \sigma \cdot \exp\left((\|\hat{p_\sigma}\| - \hat{\chi}_N)/(d_\sigma \cdot \hat{\chi}_N) \right)$
6. $\hat{p}_m = (1 - c_m) \cdot p_m + c_m^u \cdot (Z + Z_1 \cdot w_v \cdot v)$
7. $\hat{v} = (1 - c_v) \cdot \mathrm{sign}(\langle v, p_m \rangle) \cdot v + c_v \cdot \hat{p_m}$

where

$Z \in \mathbb{R}^N$ and $Z_1 \in \mathbb{R}$ random vectors,
$x, \hat{x} \in \mathbb{R}^N$ parent and offspring individuals,
$p_\sigma, \hat{p}_\sigma \in \mathbb{R}^N$ parent and offspring σ-paths,
$c_\sigma > 0$ σ-path constant and $c_\sigma^u = \sqrt{c_\sigma(2 - c_\sigma)}$, choose e.g. $c_\sigma = 4/(N + 4)$,
$\sigma, \hat{\sigma} > 0$ parent and offspring mutation step lengths,
$p_m, \hat{p}_m \in \mathbb{R}^N$ parent and offspring paths,
$c_m > 0$ path constant and $c_m^u = \sqrt{c_m(2 - c_m)}$, choose e.g. $c_m = 4/(N + 4)$,
$v, \hat{v} \in \mathbb{R}^N$ parent and offspring main vectors,
$c_v > 0$ main vector constant, choose e.g. $c_v = 2/(N + \sqrt{2})^2$,
$\hat{\chi}_N = \mathrm{E}(\|N(0, I)\|) = \sqrt{2} \cdot \Gamma(\frac{n+1}{2})/\Gamma(\frac{n}{2}) \approx \sqrt{N - \frac{1}{2}}$ (we prefer this approximation to the approximation from [1]).

The suggestions for the parameters c_σ, c_m and c_v are the same as the respective suggestions for the CMA in [1], they are good also for MVA. However, for some test functions, a greater value c_v yields a faster convergence, e.g. $c_v = 0.1$.

For the recombination, we restrict here to a simple intermediate recombination that is executed by computing the mean of the object variables, paths, step sizes, and main vectors of all participating individuals. Clearly, other recombination types are possible as well, e.g. discrete or generalized intermediate recombination.

5 Experimental Results

The MVA-ES has been tested against the CMA-ES and a standard ES with derandomized step length control. We used the test functions $f_1, \ldots, f_9$ from [1], which are nonlinear and resistent to simple hill-climbing. In addition, we test the lens optimization f_{10}. In order to obtain non-separability for $f_1, \ldots, f_9$, one determines a random orthonormal basis U before every ES run and minimizes $f_k(U \cdot x)$ instead of $f_k(x)$. Comparing the results to the case $U = I$ shows that each of the tested algorithms is invariant against any rotation of the search space. This was of course expected.

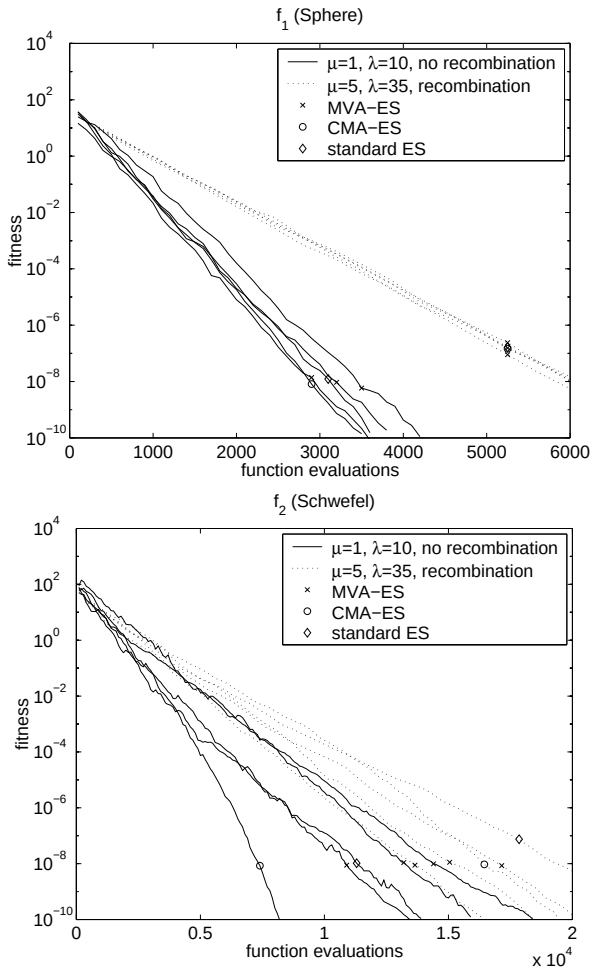


Figure 2: f_1 (sphere) and f_2 (Schwefel)

In order to compare the adapation properties, all functions have been tested in dimension $N = 20$. Moreover, we obtain a time and space complexitiy comparison with function f_1 in different dimensions. For each test function and each ES, we try a simple $(1, 10)$ variant without recombination (solid line in the plots) and a (5,35) variant with intermediate recombination (dotted line). We performed 70 runs for each setting.

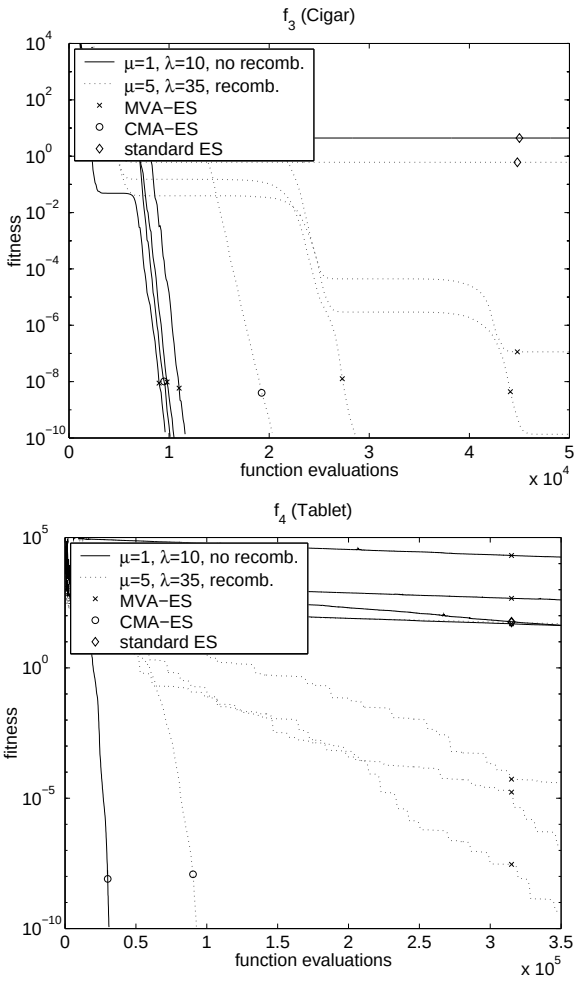


Figure 3: f_3 (cigar) and f_4 (tablet)

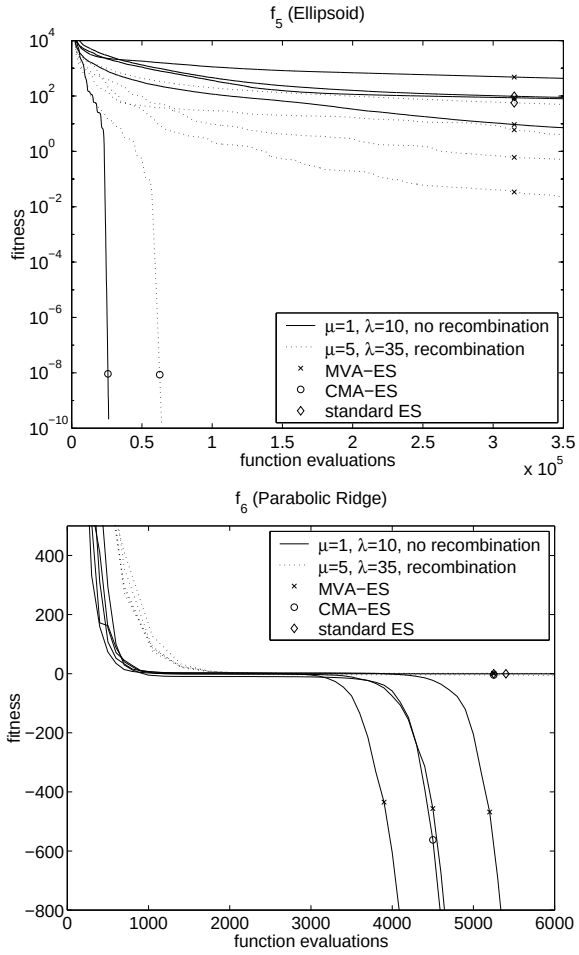


Figure 4: f_5 (ellipsoid) and f_6 (parabolic ridge)

The plots show the fitness curves of average runs of MVA-ES, CMA-ES and standard ES. For MVA-ES, the best and the worst fitness curves are displayed, too. The tests have been performed with MATLAB, for the CMA-ES we used the implementation from [1].

Function $f_1(x) = \sum_{i=1}^{N} x_i^2$ is the sphere function and the only one that remains separable under the random rotation. We observe that neither ES has difficulties to find the optimum, as well as for Schwefels function $f_2(x) = \sum_{i=1}^{N}(\sum_{j=1}^{i} x_j^2)$. The "cigar" $f_3(x) = x_1^2 + \sum_{i=2}^{N}(1000x_i)^2$ is more interesting: The standard ES fails, while CMA and MVA are successful. This is the classical case of one preferred mutation direction: It is easy to optimize the coordinates $2 \dots N$, but then the remaining feasible direction along the first coordinate is difficult to find. We observe further that recombination apparantly disturbs the main vector adaptation a little in general.

The "tablet" $f_4(x) = (1000x_1)^2 + \sum_{i=2}^{N} x_i^2$ is in a sense the converse of f_3: The optimization is first carried out along the first coordinate, then the coordinates $2 \dots N$ remain. Since there is no preferred mutation direction, it is not unexpected that MVA fails to converge, while the covariance matrix adapts easily to this situation. Here a mechanism that "fades out" one direction, i.e. the inverse of MVA could be suitable. Note that recombination helps the MVA in this case to converge. The ellipsoid $f_5(x) = \sum_{i=1}^{N}(1000^{\frac{i-1}{N-i}} x_i)^2$ is another linear transformation of the sphere. Here, there is neither a preferred mutation direction as in f_3 nor an "anti-mutation" direction as in f_4. Again, CMA adapts easily, while the bad scaling remains a problem for MVA and standard ES.

The parabolic ridge $f_6(x) = -x_1 + 100 \sum_{i=2}^{N} x_i^2$ is an instance for a preferred mutation direction and is easily optimized by the MVA-ES. The same is true for the

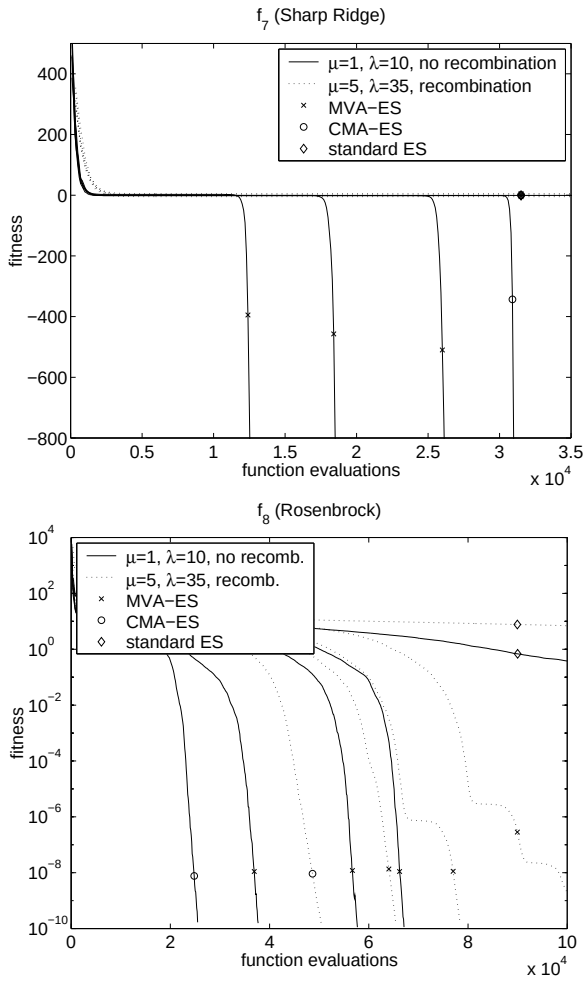


Figure 5: f_7 (sharp ridge) and f_8 (Rosenbrock)

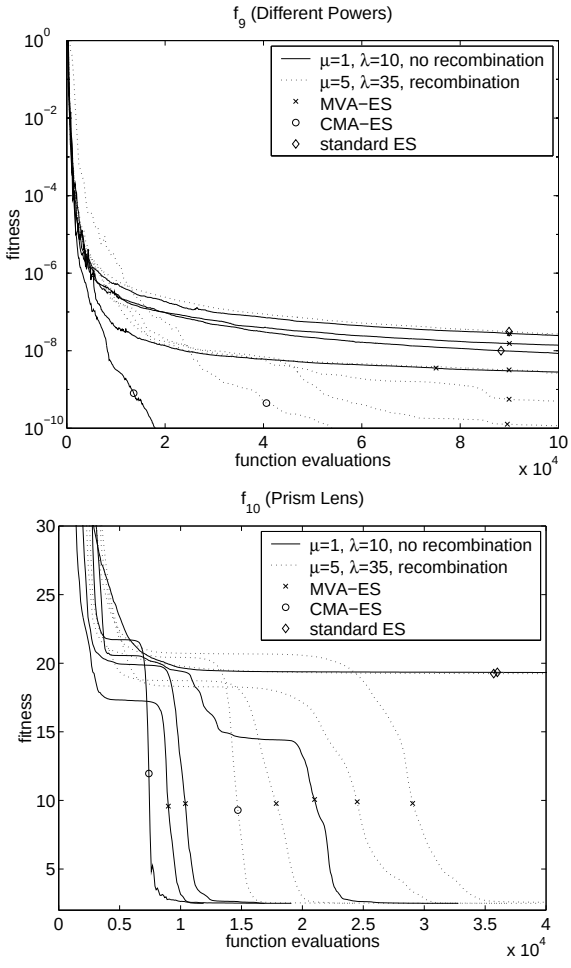


Figure 6: f_9 (different powers) and f_{10} (prism lens)

sharp ridge $f_7(x) = -x_1 + 100\sqrt{\sum_{i=2}^{N} x_i^2}$. This function is particularly hard to optimize, since the local gradient is constant. A too small choice for w_v results in a failure of the MVA. The generalized Rosenbrock function $f_8(x) = \sum_{i=1}^{N-1} \left(100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2\right)$ ("banana function") is an instance for a bent ridge. Again, MVA and CMA are successful in this situation, while standard ES fails. On the contrary, function $f_9(x) = \sum_{i=1}^{N} |x_i|^{2+10\frac{i-1}{N-1}}$, a sum of different powers, is hard for MVA. Recombination improves the MVA convergence.

Function $f_{10}(x) = \sum_{i=1}^{N-1} \left(R - \frac{h}{2} - h \cdot (i-1) - \frac{b}{h} \cdot (\epsilon - 1)(x_{i+1} - x_i)\right)^2 + \max_i x_i + \min_i x_i$ is the prism lens function (see introduction and Fig. 1). Here, $h > 0$ is the height of the segments, $b > 0$ is the distance from the lens to the screen, $R = h \cdot \frac{N-1}{2}$ the y-coordinate of the desired focal point, $\epsilon > 1$ the refraction index of the lens and $2 \cdot x_i$ the respective thickness. While the standard ES stagnates with a thickness induced by the initial random initialization, MVA and CMA find the optimum. When the preferred direction has been found and the lens is thinned, the focus is slightly disturbed and has to be restored afterwards. The MVA-ES does this much more rapidly than the CMA-ES, when $c_v = 0.1$ is chosen. (The corresponding parameter setting for CMA does not work.)

Finally, we compare the time and space consumption of a $(1, 10)$-MVA-ES and a $(1, 10)$-CMA-ES in search space dimension $N = 2, 5, 10, 20, 50, 100, 200, 400, 800$, see Fig. 7. In the time complexity plot, the average time for one generation with test function f_1 is displayed. The covariance matrix eigenvectors have been updated all $N/10$ generations. The performance decrease is linear in N for the MVA and quadratic in N for the CMA. Of course, the time complexity argu-

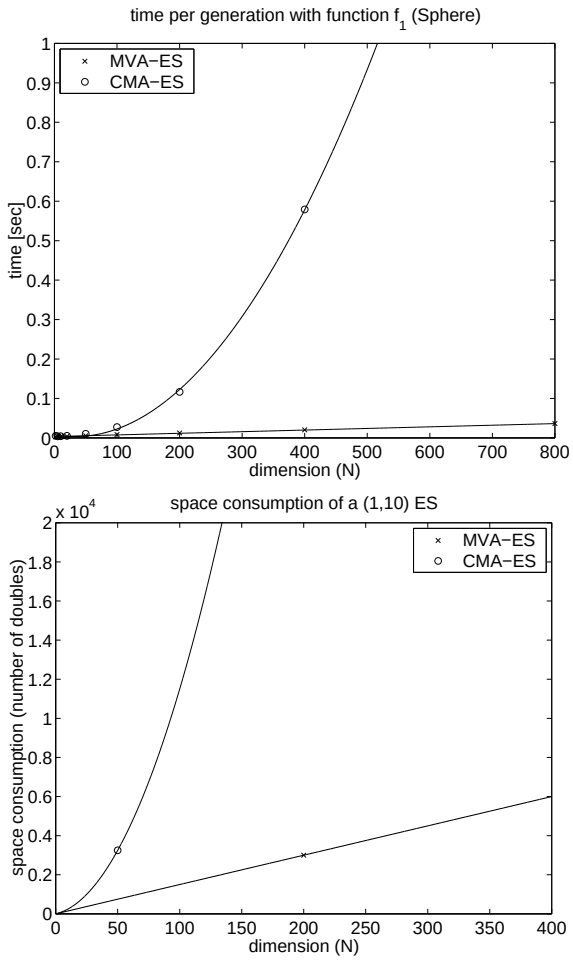


Figure 7: Time and space consumption of the $(1, 10)$-MVA-ES and the $(1, 10)$-CMA-ES for increasing dimension N. The time for one generation with test function f_1 is shown, the covariance matrix eigenvectors have been updated all $N/10$ generations.

ment becomes unimportant when the fitness function is expensive, in particular when its time complexity in N is greater or equal $O(N^2)$. But even then, the space advantage of the MVA remains, especially if the population consists of many individuals. Moreover, for large N, the computation time for the covariance matrix eigenvectors increases drastically in practice, since there is only a limited amount of memory available. For a Pentium III with 128 MB RAM and the built-in MATLAB function, this occurs at about $N \approx 400$.

6 Conclusions

We presented a new self adaptation mechanism for Evolution Strategies that adapts the main mutation vector. The algorithm is similar to CMA and shows a similar performance in situations where there is one preferred mutation direction to find. If the demanded adaptation is more complex, MVA is less powerful than CMA. On the other hand, the time and space complexity of MVA is only linear in the search space dimension N. Therefore, MVA is appropriate for problems in high dimensional search spaces ($N > 500$), where the use of CMA becomes problematic because of its $O(N^2)$ complexity. In low dimensions ($N < 100$), CMA will remain the better choice.

There are several possible extensions of MVA. For example, one could adapt an "anti-mutation" vector, this can be appropriate for functions similar to f_4. One can adapt more than one main vector, controlled e.g. by the scalar product. If this is extended to N vectors, it could be possible to obtain an algorithm similar to CMA that does not need any eigenvector decomposition and thus has an $O(N^2)$ update of the mutation directions instead of $O(N^3)$.

Acknowledgments.

We would like to thank Nikolaus Hansen for providing us with the CMA-ES code. Furthermore, we thank Jürgen Wakunda and Kosmas Knödler for helpful discussions. This research has been supported by the BMBF (grant no. 01 IB 805 A/1).

References

[1] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. Preprint 2000, to appear in: Evolutionary Computation, Special Issue on Self-Adaptation.

[2] N. Hansen and A. Ostermeier. Convergence properties of evolution strategies with the derandomized covariance matrix adaptation: The $(\mu/\mu_i, \lambda)$-cma-es. In *5th European Congress on Intelligent Techniques and Soft Computing*, pages 650–654, 1997.

[3] A. Ostermeier. An evolution strategy with momentum adaptation of the random number distribution. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature II*, pages 197–206, 1992.

[4] I. Rechenberg. *Evolutionsstrategie '94*. frommann-holzboog, Stuttgart, 1994.

[5] H.-P. Schwefel. *Numerical Optimization of Computer Models*. Wiley, New York, 1995.

Fuzzy Evolutionary Algorithm for VLSI Placement

Sadiq M. Sait **Habib Youssef** **Junaid A. Khan**

Computer Engineering Department
King Fahd University of Petroleum and Minerals
KFUPM Box # 673, Dhahran-31261, Saudi Arabia
e-mail: {sadiq,youssef,jakhan}@ccse.kfupm.edu.sa
Telephone: : +966-3-860-6665, Fax: : +966-3-860-3059.

Abstract

This paper presents a Fuzzy Simulated Evolution (FSE) algorithm for VLSI standard cell placement. This is a hard multiobjective combinatorial optimization problem with no known exact and efficient algorithm that can guarantee finding a solution of specific or desirable quality. Approximation iterative heuristics such as Simulated Evolution are best suited to perform an intelligent search of the solution space. Due to the imprecise nature of design information at the placement stage the various objectives and constraints are expressed in the fuzzy domain. The search is made to evolve toward a vector of fuzzy goals. The proposed heuristic is compared with genetic algorithm. FSE was able to achieve better solutions than GA in a fraction of the time.

1 INTRODUCTION

In VLSI design, the placement problem consists of assigning modules to locations on the silicon surface under numerous design constraints while trading-off several objectives. The number of modules can be in range of thousands. Even in its simplest form, placement is a generalization of the quadratic assignment problem [7].

Formally this problem can be stated as follows: Given a set of modules $M = \{m_1, m_2, \cdots, m_n\}$, and a set of signals $V = \{v_1, v_2, \cdots, v_k\}$, each module $m_i \in M$ is associated with a set of signals V_{m_i}, where $V_{m_i} \subseteq V$. Also each signal $v_i \in V$ is associated with a set of modules M_{v_i}, where $M_{v_i} = \{m_j | v_i \in V_{m_j}\}$. M_{v_i} is called a signal net. Placement problem is to assign each module $m_i \in M$ to a unique location such that a given cost function is optimized and constraints are satisfied. Objectives addressed in this work are the minimization of wire-length, power dissipation, and circuit delay. Layout area is considered as a constraint. These are estimated as follows.

Estimation of Wire-length: The wire-length cost can be computed by adding wire-length estimates for all the nets in the circuit.

$$\text{Cost}_{wire} = \sum_{j \in M} l_j \tag{1}$$

where l_j is the wire-length associated with net v_j and M is the set of all cells in the circuit. This wire-length is computed using Steiner tree approximation.

Estimation of Power: In CMOS circuits, over 90% power dissipation is due to the switching activity [2, 1], expressed as:

$$P_t \simeq \sum_{i \in M} \frac{1}{2} \cdot C_i \cdot V_{DD}^2 \cdot f \cdot S_i \cdot \beta \tag{2}$$

where P_t denotes the total power, V_{DD} is the supply voltage, S_i is the switching activity at the output node of cell i (module m_i) which indicate the number of gate output transition per clock cycle, f is the clock frequency. The node total capacitance is denoted by C_i, and β is a technology dependent constant. Assuming that clocking frequency and power voltages are fixed, total power dissipation of the circuit is a function of C_i and S_i of the various gates in the logic circuit. The capacitive load C_i of a gate comprises input gate capacitances of the cells driven by cell i and that of interconnects capacitance at the cell output node, as shown in the following equation:

$$C_i = C_i^r + \sum_{j \in M_i} C_j^g \tag{3}$$

where C_j^g is the input capacitance for cell (or gate) j, C_i^r represents the interconnect capacitance at the output node of cell i and M_i is the set of fanout cells of cell i.

In case of standard cell design, cell properties are fixed for a particular library, hence we cannot reduce C_j^g. Further C_i^r are related to the corresponding interconnect wire-lengths l_i, hence cost due to the overall power in VLSI circuits can be termed as:

$$\text{Cost}_{power} = \sum_{i \in M} S_i l_i \tag{4}$$

Estimation of Delay: Let path π consist of nets $\{v_1, v_2, ..., v_k\}$, then its path delay T_π is expressed by the following equation:

$$T_\pi = \sum_{i=1}^{k} (CD_i + ID_i) \tag{5}$$

where CD_i is the switching delay of the cell driving net vi and ID_i is the interconnect delay of net vi. The overall circuit delay is equal to T_{π_c}, where π_c is the longest path in the layout (most critical path).

CD_i is constant and only ID_i depends on placement. Using the RC delay model, this delay is estimated as:

$$ID_i = (LF_i + R_i^r) \times C_i \tag{6}$$

where LF_i is load factor of the driving block (independent of layout), R_i^r is the interconnect resistance of net v_i, and C_i is the load capacitance of cell i given in Equation 3. The cost function due to timing performance can be expressed as:

$$\text{Cost}_{delay} = T_{\pi_c} \tag{7}$$

Layout Width: In our work layout width is considered as a constraint. The upper limit on the layout width is defined in Equation 8:

$$Width_{max} = (1 + a) \times Width_{opt} \tag{8}$$

where $Width_{max}$ is the maximum allowable width of the layout, $Width_{opt}$ is the minimum possible layout width obtained by adding the widths of all cells and dividing it by the number of rows in the layout. The parameter a denotes how wide the layout can be as compared to the optimal one.

During placement most of the cost parameters cannot be precisely determined. For example, the exact amount of wire-length and area of the layout can be known only after the subsequent stage of design (routing). Also, the amount of power dissipated, or the performance of the circuit, depends on the amount of wire-length as well as the operation dynamics of the circuit. For these reasons it is much easier for a designer to describe desirable characteristics of placement solutions in linguistic terms which is the basis of fuzzy logic [11]. In this work the evaluation of the *goodness* values of individual modules in their current locations (a requirement of simulated evolution heuristic) as well as the quality of the overall placement solution are described using fuzzy rules.

2 SIMULATED EVOLUTION

Placement is a hard combinatorial optimization problem with no known exact and efficient optimization algorithms that can find a solution of a given quality. Approximation iterative heuristics such as simulated annealing, genetic algorithms, simulated evolution, tabu search, are robust search methods for this category of problems [8].

Simulated Evolution (SE) is a general iterative heuristic proposed by Ralph Kling [6]. It falls in the category of algorithms which emphasize the behavioral link between parents and offspring, or between reproductive populations, rather than the genetic link [4]. This scheme combines iterative improvement and constructive perturbation and saves itself from getting trapped in local minima by following a stochastic perturbation approach. It iteratively operates a sequence of evaluation, selection and allocation steps on one solution. The selection and allocation steps constitute a compound move from current solution to another feasible solution of the state space. SE proceeds as follows. It starts with a randomly or constructively generated valid initial solution. A solution is seen as a set of movable elements (modules). Each element m_i has an associated goodness measure g_i in the interval [0,1]. The main loop of the algorithm consists of three steps (See Figure 1): **evaluation**, **selection** and **allocation**. These steps are carried out repetitively until some stopping condition is satisfied. In the **evaluation** step, the goodness of each element is estimated. In the **selection** step, a subset of elements are selected and removed from current solution. The lower the goodness of a particular element, the higher is its selection probability. A bias parameter B is used to compensate for inaccuracies of the *goodness* measure. Finally, the **allocation** step tries to assign the selected elements to better locations. Other than these three steps, some input parameters for the algorithm are set in an earlier step known as **initialization**.

```
Algorithm Simulated_Evolution(B, Φ_initial, StoppingCondition)
NOTATION
B= Bias Value.
Φ= Complete solution.
m_i= Module i.
g_i= Goodness of m_i.
ALLOCATE(m_i, Φ_i)=Function to allocate m_i in partial solution Φ_i
Begin
Repeat

  EVALUATION:
      ForEach m_i ∈ Φ evaluate g_i;
      /* Only elements that were affected by moves of previous */
      /* iteration get their goodnesses recalculated*/
  SELECTION:
      ForEach m_i ∈ Φ DO
          begin
            IF Random > Min(g_i + B, 1)
            THEN
              begin
                S = S  ∪  m_i; Remove m_i from Φ
              end
          end
    Sort the elements of S
  ALLOCATION:
      ForEach m_i ∈ S DO
          begin
            ALLOCATE(m_i, Φ_i)
          end
Until  Stopping Condition is satisfied
Return Best solution.
End (Simulated_Evolution)
```

Figure 1: Structure of the simulated evolution algorithm.

3 FUZZY SIMULATED EVOLUTION

In order to apply simulated evolution one has to design a suitable goodness measure, a cost function, and an appropriate allocation operator. These three together have the most impact on the behavior of the SE algorithm. Due to the multiobjective nature of the placement problem, the goodness measure, cost function, and the allocation operator should take into consideration all objectives.

Balancing different objectives by weight functions is difficult, or at best controversial. Fuzzy logic is a convenient vehicle for solving this problem. It allows to map values of different criteria into linguistic values, which characterize the level of satisfaction of the designer with the numerical value of the objectives. All these numerical values operate over values from the interval [0,1] defined by the membership functions for each objective. For placement, the designer seeks to find solutions optimized with respect to *wire-length*, *delay*, and *power dissipation*.

3.1 FUZZY GOODNESS EVALUATION

Following the generation of an initial solution, the goodness of each cell in its current location is determined. A designated location of a cell is considered good if it results in short wire-length for its nets, reduced delay, and reduced power. These conflicting requirements can be conveniently expressed by the following fuzzy logic rule.

Rule R1: IF cell i is *near its optimal wire-length* AND *near its optimal power* AND (*near its optimal net delay* OR $T_{max}(i)$ *is much smaller than* T_{max}) **THEN** it has a high goodness.

where T_{max} is the delay of most critical path in the current iteration and $T_{max}(i)$ is the delay of the longest path traversing cell i in the current iteration.

A fuzzy logic rule is an **If-Then** rule. The **If** part (*antecedent*) is a fuzzy predicate defined in terms of linguistic values and fuzzy operators (**Intersection** and **Union**). The **Then** part is called the *consequent*. In our case, the linguistic value used in the consequent part identifies the fuzzy subset of good locations for that particular cell. There are many implementations of fuzzy union and fuzzy intersection operators. Fuzzy union operators are known as **s-norm** operators while fuzzy intersection operators are known as **t-norm**. Generally, **s-norm** is implemented using max and **t-norm** as min function, i.e., $\mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x))$, and $\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x))$. This is known as the $\min - \max$ logic initialy introduced by Zadeh [11]. For example, according to the min-max logic the rule above evaluates to the following:

$$\mu_i^e(x) = \min\left(\mu_{iw}^e(x), \mu_{ip}^e(x), \max(\mu_{inet}^e(x), \mu_{ipath}^e(x))\right) \quad (9)$$

where the superscript e is used here to represent evaluation, x represents the location of cell i, $\mu_i^e(x)$ represents the membership in the fuzzy set of high goodness, $\mu_{iw}^e(x)$ and $\mu_{ip}^e(x)$ represent the memberships in fuzzy subsets of *near optimal wire-length* and *low power* as compared to other cells; $\mu_{inet}^e(x)$ and $\mu_{ipath}^e(x)$ are the memberships in fuzzy sets of *near optimal net delay* as compared to other cells and "$T_{max}(i)$ is *much smaller than* T_{max}".

However, formulation of multi-criteria decision functions do not desire pure "anding" of **t-norm** nor the pure "oring" of **s-norm**. The reason for this is the complete lack of compensation of **t-norm** for any partial fulfillment and complete submission of **s-norm** to fulfillment of any criteria. Also the indifference to the

individual criteria of each of these two forms of operators led to the development of Ordered Weighted Averaging (OWA) operators [9]. This operator falls in the category of compensatory fuzzy operators and allows easy adjustment of the degree of "anding" and "oring" embedded in the aggregation. According to [9], "orlike" and "andlike" OWA for two fuzzy sets A and B are implemented as given in Equations 10 and 11 respectively.

$$\mu_{A \bigcup B}(x) = \beta \times \max(\mu_A, \mu_B) + (1-\beta) \times \frac{1}{2}(\mu_A + \mu_B) \quad (10)$$

$$\mu_{A \bigcap B}(x) = \beta \times \min(\mu_A, \mu_B) + (1-\beta) \times \frac{1}{2}(\mu_A + \mu_B) \quad (11)$$

β is a constant parameter in the range [0,1]. It represents the degree to which OWA operator resembles the pure "or" or pure "and" respectively.

With the AND and OR fuzzy operators implemented as OWA operators, rule **R1** evaluates to the expression below:

$$g_i = \mu_i^e(x) = \beta^e \times min(\mu_{iw}^e(x), \mu_{ip}^e(x), \mu_{id}^e(x)) + (1-\beta^e) \times \frac{1}{3} \sum_{j=w,p,d} \mu_{ij}^e(x) \quad (12)$$

where

$$\mu_{id}^e(x) = \beta_d^e \times max(\mu_{inet}^e(x), \mu_{ipath}^e(x)) + (1-\beta_d^e) \times \frac{1}{2}(\mu_{inet}^e(x) + \mu_{ipath}^e(x)) \quad (13)$$

g_i is the goodness of cell i. β^e and β_d^e are constants between 0 and 1 to control OWA operators. Whereas $\mu_{id}^e(x)$ represents the membership in fuzzy set of *good timing performance*, it is obtained after applying orlike OWA to $\mu_{inet}^e(x)$ and $\mu_{ipath}^e(x)$.

$\mu_{ipath}^e(x)$ is included in the computation of $\mu_{id}^e(x)$ because if a cell is not on the critical path then it must have high goodness with respect to the delay objective.

If a cell i drives the net v_i, $\{v_1, v_2, ..., v_k\}$ is the set of nets connected to cell i and v_p is the net driven by the predecessor cell of cell i on the longest path traversing cell i, then base values $X_{iw}(x)$, $X_{ip}(x)$, $X_{inet}(x)$ for fuzzy sets near optimal wire-length, power, net delay and $X_{ipath}(x)$ for fuzzy set "$T_{\max}(i)$ much smaller than $T_{\max}$" are computed as given in Equations 14-17,

$$X_{iw}(x) = \frac{\sum_{j=1}^{k} l_j^*}{\sum_{j=1}^{k} l_j} \quad (14)$$

$$X_{ip}(x) = \frac{\sum_{j=1}^{k} S_j \, l_j^*}{\sum_{j=1}^{k} S_j \, l_j} \quad (15)$$

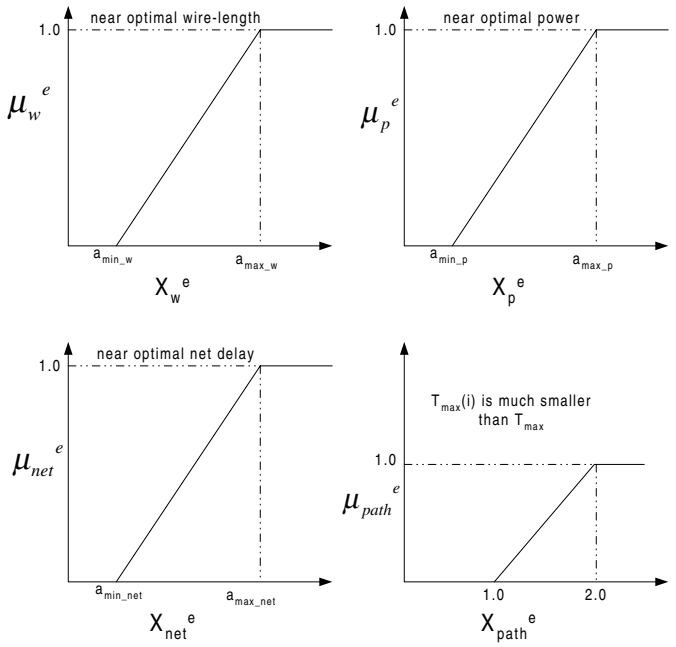


Figure 2: Membership functions used in fuzzy evaluation.

$$X_{inet}(x) = \frac{ID_i^* + ID_{ip}^*}{ID_i + ID_{ip}} \quad (16)$$

$$X_{ipath}^e(x) = \frac{T_{max}}{T_{max}(i)} \quad (17)$$

where l_j^* is the optimal wire-length of net v_j, computed by placing all the cells connected to v_j next to each other on the layout surface and then estimating the wire-length; the product $S_j \times l_j$ is related to the switching power dissipated in net v_j; ID_i^* is the optimal interconnect delay of net v_i, ID_{ip} and ID_{ip}^* are the actual and optimal interconnect delays of net driven by the predecessor cell of cell i on the current longest path traversing cell i. Membership functions of these base values are shown in Figure 2.

The values of a_{min} and a_{max} depend on the statistical nature of the base values. A typical frequency of occurance plot is shown in Figure 3, where we have plotted $X_{net}^e(x)$ and $X_w^e(x)$ versus the number of cells having these base values. It is clear from this figure that these plots have nearly bell-shaped behavior. Therefore we can say that around 95% cells have base values in the range $[\bar{X}_i - 2\sigma_i, \bar{X}_i + 2\sigma_i]$, where $\bar{X}_i$ is the mean value of $X_i(x)$ and σ_i is the standard deviation of $X_i(x)$ for $i = w, p, net$. The values of a_{min} and a_{max} are therefore computed as:

$$a_{min\_i} = \bar{X}_i - 2\sigma_i \quad and \quad a_{max\_i} = \bar{X}_i + 2\sigma_i \quad (18)$$

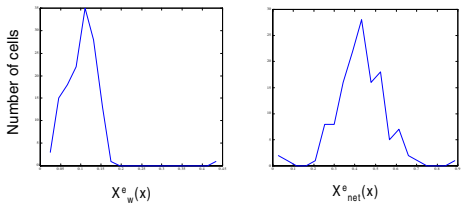


Figure 3: Base values vs frequency of occurance plot.

The values of a_{min} and a_{max} are computed in the beginning and then recomputed again when the size of selection set is around 90 percent of the initial value.

3.2 SELECTION

In this stage of the algorithm some cells are selected probabilistically depending on their goodness values. Bias concept in selection step, present in the original SE algorithm [5], is the major drawback of the heuristic. It is not easy to select value of bias because it varies from problem to problem. Also in the case of placement it varies from circuit to circuit [10]. To overcome this problem another selection scheme is proposed. According to this scheme a random number is generated in the range $[0, M]$ and compared with g_i. If the generated random number is greater than g_i then the cell is selected for allocation. The value of M is calculated as follows:

$$M = \bar{G} + 2\sigma_g \tag{19}$$

where $\bar{G}$ and σ_g are the average goodness and standard deviation of goodness values of cells in current iteration. Value of M is calculated in the beginning and updated only once, when the size of selection set is 90% of its initial size.

3.3 ALLOCATION

In allocation stage the selected cells are to be placed in the best available locations. In our proposed scheme we have considered selected cells as movable modules and remaining cells as fixed modules. Selected cells are sorted in descending order of their goodnesses with respect to their partial connectivity with unselected cells. Ties are broken with respect to their goodness values. One cell from the sorted list is selected at a time and its location is swapped with other movable cells in the current solution. The swap that results in the maximum gain is accepted and the cell is removed from the selection set.

The goodness of the new location is characterized by the following fuzzy rule:

Rule R2: IF a swap results in *reduced overall wire-length* AND *reduced overall power* AND *reduced delay* AND *within acceptable layout width* **THEN** it gives good location.

The above rule is interpreted as follows.

$$\mu^a_{i\_pwd}(l) = \beta^a \times min(\mu^a_{ip}(l),\ \mu^a_{iw}(l),\ \mu^a_{id}(l)) + (1-\beta^a) \times \frac{1}{3} \sum_{j=p,w,d} \mu^a_{ij}(l) \tag{20}$$

$$\mu^a_i(l) = min(\ \mu^a_{i\_width}(l),\ \mu^a_{i\_pwd}(l)\ ) \tag{21}$$

the superscript a is used here to represent allocation. $\mu^a_i(l)$ is the membership of cell i at location l in the fuzzy set of good location. $\mu^a_{i\_pwd}(l)$ is the membership in the fuzzy set of "reduced wire-length and reduced power and reduced delay". $\mu^a_{iw}(l)$, $\mu^a_{ip}(l)$, $\mu^a_{id}(l)$, and $\mu^a_{iwidth}(l)$ are the membership in the fuzzy sets of reduced wire-length, reduced power, reduced delay and within acceptable width respectively.

Notice that the third AND operator in the above fuzzy rule is implemented as a pure min because the width constraint has to be always satisfied.

If a cell i swaps its location with cell j then the base values are computed as shown in Equations 22-25:

$$X^a_{iw}(l) = \frac{(\sum_{m=1}^{k_i} l_{im} + \sum_{m=1}^{k_j} l_{jm})_n}{(\sum_{m=1}^{k_i} l_{im} + \sum_{m=1}^{k_j} l_{jm})_{n-1}} \tag{22}$$

$$X^a_{ip}(l) = \frac{(\sum_{m=1}^{k_i} S_{im} l_{im} + \sum_{m=1}^{k_j} S_{jm} l_{jm})_n}{(\sum_{m=1}^{k_i} S_{im} l_{im} + \sum_{m=1}^{k_j} S_{jm} l_{jm})_{n-1}} \tag{23}$$

$$X^a_{id}(l) = \frac{(ID_i + ID_{ip} + ID_j + ID_{jp})_n}{(ID_i + ID_{ip} + ID_j + ID_{jp})_{n-1}} \tag{24}$$

$$X^a_{i\_width}(l) = \frac{Width_n}{Width_{opt}} \tag{25}$$

where, subscript n and $n-1$ show the iteration numbers, $\{v_{i1}, v_{i2}, ..., v_{ik_i}\}$ is the set of nets connected to cell i, $Width_n$ is the actual width at n^{th} iteration.

Membership functions for these base values are shown in Figure 4. The values of a_w, a_p, a_d and a_{width} depend upon priority on the optimization level of the respective objective. Typical values for a_w, a_p and a_d are in the range $[0.75, 0.95]$, whereas a_{width} is in the range $[0.2, 0.5]$. In our case we have set $a_w = 0.75$, $a_p = 0.75$, $a_d = 0.85$ and $a_{width} = 0.25$.

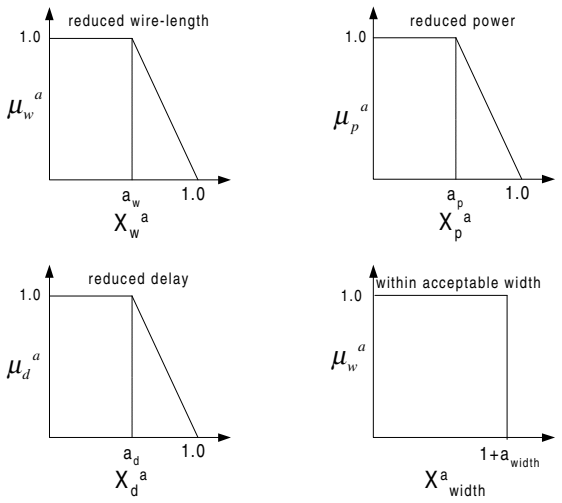


Figure 4: Membership functions used in allocation.

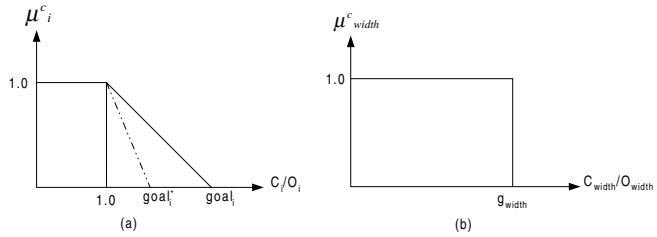


Figure 5: Membership functions within acceptable range.

3.4 FUZZY COST MEASURE

In order to select the best solution found so far, it is required to develop some cost measure. In case of multi-objective placement, the best placement is one which results in lowest cost in terms of all objectives. However, such a solution most likely does not exist. Some techniques to cope with this problem are mentioned in [12]. All these techniques introduce some tradeoff between different objectives. In this work, a goal directed search approach is adopted, where the best placement is one that satisfies as much as possible a user specified vector of fuzzy goals [3].

In order to combine three parameters and one constraint, following fuzzy rule is suggested.

Rule R3: IF a solution is within *acceptable wire-length* AND *acceptable power* AND *acceptable delay* AND *within acceptable layout width* **THEN** it is an acceptable solution.

The above fuzzy rule translates to the following:

$$\mu^c_{pdl}(x) = \beta^c \times min(\mu^c_p(x), \mu^c_d(x), \mu^c_l(x)) + (1-\beta^c) \times \frac{1}{3} \sum_{j=p,d,l} \mu^c_j(x) \quad (26)$$

$$\mu^c(x) = min(\mu^c_{pdl}(x), \mu^c_{width}(x)) \quad (27)$$

where $\mu^c(x)$ is the membership of solution x in fuzzy set of acceptable solutions, $\mu^c_{pdl}(x)$ is the membership in fuzzy set of "acceptable power AND acceptable delay AND acceptable wire-length", whereas $\mu^c_j(x)$ for $j = p, d, l, width$, are the individual membership values in the fuzzy sets *within acceptable wire-length, power, delay, and layout width* respectively. β^c is the constant in the range $[0, 1]$, in our case we chose $\beta^c = 0.7$, the superscript c represents "cost". The solution that results in maximum value of $\mu^c(x)$ is reported as the best solution found by the search heuristic.

The membership functions for fuzzy sets *within acceptable power, delay and wire-length* are shown in Figure 5(a), whereas the constraint *within acceptable layout width* is given as a crisp set as shown in Figure 5(b).

Since layout width is a constraint, hence its membership value is either 1 or 0 depending upon $goal_{width}$ (in our case $goal_{width} = 1.25$). However, for other objectives by increasing and decreasing the value of $goal_i$ we can vary its preference in overall membership function. The lower bounds ($O_i s$ for $i \in \{l, p, d, width\}$) are computed as follows: $O_l = \sum_{i=1}^{n} l^*_i$, $O_p = \sum_{i=1}^{n} S_i l^*_i, \forall v_i \in \{v_1, v_2, ..., v_n\}$; $O_d = \sum_{j=1}^{k} CD_j + ID^*_j \ \forall v_j \ in \ path \ \pi_c$; and $O_{width} = Width_{opt}$; where k is the number of nets in π_c.

4 GA BASED OPTIMIZATION

For the comparison purpose we have also implemented **genetic algorithm (GA)** [8]. Membership value in the fuzzy set of acceptable solution given in Equation 27 is used as the **fitness** measure of a chromosome (solution). In parent selection step for crossover **roulette wheel** selection scheme [8], is used. **Partially mapped crossover (PMX)** is used to generate new offsprings. For the selection of next generation **Extended Elitism Random Selection** scheme is used, where half of the chromosomes in the next population are the best among offspring and current population and half are selected randomly. A variable **mutation** is used in the range $[0.03, 0.05]$ that depends upon the standard deviation of fitness in the current population. Stopping criteria is the maximum number of generations.

5 EXPERIMENTS AND RESULTS

We have applied GA and FSE on eleven different ISCAS-85 and ISCAS-89 benchmark circuits. In case of FSE algorithm, execution is aborted when no improvement is observed in the best solution found so far in 500 consecutive iterations. In case of GA stopping criteria is 10,000 generations.

Table 1 compares the quality of final solution generated by FSE and GA. The circuits are listed in order of their size (122-1753 modules). From the results, it is clear that GA performs better than FSE for smaller circuits, but for circuits with large number of cells FSE outperforms GA. In all circuits it is observed that GA takes considerably large amount of execution time as compared to FSE.

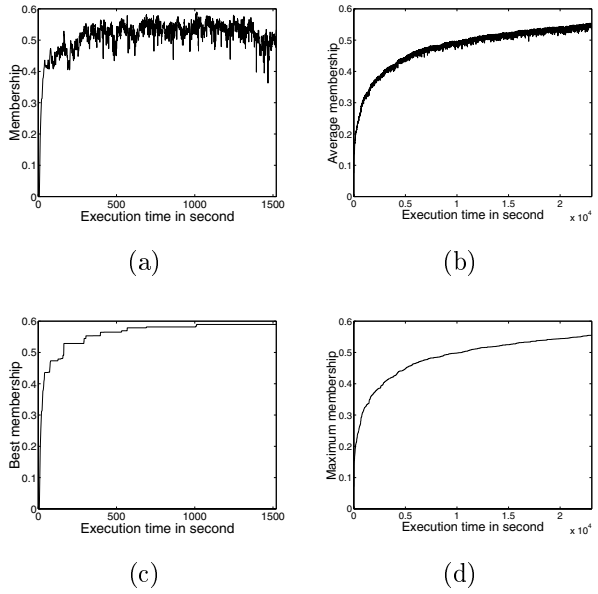


Figure 6: Comparison of FSE and GA. (a) and (c) represent current and best fitness (membership) of the solution obtained by FSE; (b) and (d) represent average and best solution obtained by GA, plotted versus execution time in seconds.

To compare improvement in the quality of solution versus time, we have plotted current and best membership values of the solution obtained by FSE versus actual execution time in Figure 6-(a) and (c), for comparison the average and best fitness (membership) values in a current population obtained by GA versus execution time are plotted in Figure 6-(b) and (d). These plots are for test case S1196. It can be observed that quality of solution improves rapidly in FSE based search as compared to GA. Due to lack of space plots for other circuits are not included; however both heuristics exhibited similar behavior on all circuits.

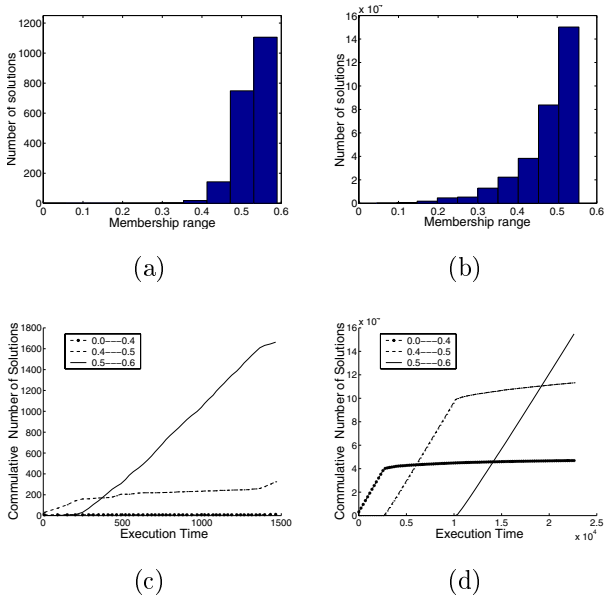


Figure 7: Comparison of FSE and GA based on search efforts in particular membership ranges; (a) and (b) show number of solutions visited in particular membership ranges for FSE and GA respectively; (c) and (d) show cumulative number of solutions visited in a specific membership range versus execution time in seconds for FSE and GA respectively.

Figures 7(a) and (b) show the quality of solution subspace searched by FSE and GA. It is evident from the figure that both heuristics concentrated in high fitness subspaces which indicates that they were properly engineered to solve this particular problem. The figure also shows that FSE was able to evolve much faster toward a better solution subspace (after few hundred seconds). On the other hand GA required generations in the order of thousands, where each generation consisted of 32 solutions.

Figures 7(c) and (d) track with time the total number of solutions found by FSE and GA for various membership ranges. These are very informative plots as they show, that as time progressed, the solutions found by each heuristic were getting better. Note however that FSE exhibited much faster evolutionary rate than GA. For example, after about 400 seconds, almost all new solutions discovered by FSE have a membership in the interval 0.5-0.6 in the fuzzy subset of good solutions with respect to all objectives, and almost none were found with lower membership values (see Figure 7(c)). In contrast, for GA, it is only after 10,000 seconds that the first solution with membership in the interval 0.5-0.6 was found (see Figure 7(d)). This behavior was observed for all test cases.

Circuit	GA					FSE				
	L (μm)	P (μm)	D (ps)	W (μm)	T (s)	L (μm)	P (μm)	D (ps)	W (μm)	T(s)
S2081	2426	388	113	142	2341	2693	462	112	152	43
S298	4062	838	130	171	2922	4989	1013	133	181	104
S386	6824	1665	193	181	3945	7088	1640	197	186	152
S832	21015	4787	395	232	7206	24705	5827	390	258	1643
S641	18320	4365	736	254	21982	13906	3321	702	296	618
S953	32031	5156	230	262	11221	32340	5242	245	262	1278
S1238	52679	15473	410	279	16208	39629	12377	371	310	1168
S1196	51804	15259	370	292	23070	42426	12745	364	325	1521
S1494	71021	17497	803	336	26032	56961	14071	719	360	3378
S1488	69792	17346	784	334	21434	57091	13887	710	358	3529
C3540	310996	109850	924	427	43232	164897	58055	734	507	18318

Table 1: Layout found by FSE and GA, "L", "P" "D" and "W" represent the wire-length, power, delay, and width costs and "T" represents execution time in seconds

6 CONCLUSION

In this paper, we have proposed an evolutionary algorithm for low power high performance VLSI standard cell placement. We have used FSE as search heuristic and GA for comparison. Fuzzy logic is used to overcome the multi-objective nature of the problem. In FSE, fuzzy logic was implemented at evaluation and allocation stages and to choose the best solution from the set of solutions generated by FSE. In GA fuzzy logic is used in the fitness evaluation.

It is observed that FSE performs much better than GA in terms of execution time, it also performs better than GA in terms of final solution in bigger circuits. Also the quality of solution improved more rapidly in FSE based search as compared to GA.

Acknowledgments

Authors acknowledge King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for all support. This work is carried out under university funded project number COE/ITERATE/221.

References

[1] A. Chandrakasan, T. Sheng, and R. W. Brodersen. Low Power CMOS Digital Design. *Journal of Solid State Circuits*, 4(27):473–484, April 1992.

[2] Srinivas Devadas and Sharad Malik. A Survey of Optimization Techniques Targeting Low Power VLSI Circuits. *32nd ACM/IEEE Design Automation Conference*, 1995.

[3] Henrik Esbensen and Ernest S. Kuh. Design space exploration using the genetic algorithm. *ISCAS '96. IEEE International Symposium on Circuits and Systems*, pages 500–503, 1996.

[4] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, Jan 1994.

[5] R. M. Kling and P. Banerjee. ESP: Placement by Simulated Evolution. *IEEE Transaction on Computer-Aided Design*, 3(8):245–255, March 1989.

[6] Ralph Michael Kling. *Optimization by Simulated Evolution and its Application to cell placement.* Ph.D. Thesis, University of Illinois, Urbana, 1990.

[7] Sadiq M. Sait and Habib Youssef. VLSI Physical Design Automation: Theory and Practice. *Mc Graw-Hill Book Company, Europe*, 1995.

[8] Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems.* IEEE Computer Society Press, California, December 1999.

[9] Ronald R. Yager. On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transaction on Systems, MAN, and Cybernetics*, 18(1), January 1988.

[10] Habib Youssef, Sadiq M. Sait, and Ali Hussain. Adaptive Bias Simulated Evolution Algorithm for Placement. *IEEE International Symposium on Circuits and Systems*, May 2001.

[11] L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transaction Systems Man. Cybern*, SMC-3(1):28–44, 1973.

[12] Eckart Zitzler. Evolutionary optimization for multiobjective optimization: Methods and applications. *DTS thesis, Swiss Federal Institute of Technology Zurich*, November 1999.