

MACHINE SCIENCE:  
AUTOMATED MODELING OF DETERMINISTIC AND STOCHASTIC  
DYNAMICAL SYSTEMS

A Dissertation  
Presented to the Faculty of the Graduate School  
of Cornell University  
In Partial Fulfillment of the Requirements for the Degree of  
Doctor of Philosophy

by  
Michael Douglas Schmidt  
January 2011

© 2011 Michael Douglas Schmidt

MACHINE SCIENCE:  
AUTOMATED MODELING OF DETERMINISTIC AND STOCHASTIC  
DYNAMICAL SYSTEMS

Michael Douglas Schmidt, Ph. D.

Cornell University 2011

The work presented here advances the technology to analyze experimental data and automatically hypothesize about explanatory models and physical laws that help explain observations. Automated Modeling, sometimes referred to as Symbolic Regression or System Identification, is the process of searching a possibly infinite space of mathematical expressions in order to optimize various objectives – for example, identifying the simplest possible nonlinear equation that captures the observed dynamics of a system.

Traditionally, the task of formulating analytical models and theory has remained entirely within the purview of human expertise, and also human limitation. However, the development of Evolutionary Algorithms, and more recently Genetic Programming, has made searching for analytical models automatically a possibility. The work presented here focuses on advancing the algorithms and techniques for Automated Modeling to shrink this “reality gap,” and applies these advances to various real and experimental systems for the first time.

The specific contributions of this work fall into four categories: search methods and algorithms, model representations and the types of systems that can be analyzed,

techniques for interpreting solutions and results, and applications in science and engineering fields.

The most important contribution in the search methods is the Fitness and Rank Prediction algorithm, which enables utilizing exceedingly large data sets with low computational effort. This algorithm is based on the idea that, at any given time, only a small number of carefully selected data points are necessary to discriminate among candidate models, allowing large reductions in computational effort. In model representations, the most important contribution is the principle for identifying meaningful invariant quantities amongst the infinite number of trivial invariant expressions. This principle enables searching for physical laws and conservations directly from experimental measurements. In the interpretation of results, the most important contribution is Parameter Mapping technique, which relates an automatically inferred model to a previous model through repeated regressions. Finally, the most important contribution in applications is the analysis of yeast Glycolytic oscillations, which demonstrates and compares several techniques in order to identify a complete nonlinear ordinary differential equation model directly from data.

## BIOGRAPHICAL SKETCH

Michael Schmidt was born in La Crosse, Wisconsin on December 6, 1981 to Mary and Douglas Schmidt. His mother was a school teacher who later started a direct mailing business. His father was a mechanical engineer who later started a software company. Michael's mother introduced him to art and music at an early age and encouraged many creative pursuits throughout his childhood – particularly painting and playing piano. Michael's father first introduced him to engineering and mathematics – for example, teaching him to estimate the distance a model rocket would drift when lost to the wind. Michael went to public school as a child. In secondary school, he tutored math and physics students, and represented the school in a few academic competitions. He also played baseball and hockey, but his primary interest eventually turned to computers. Programming became Michael's creative outlet as the internet burgeoned in the 1990s. He taught himself to program with his best friend by working on several nefarious projects – especially, writing tools to exploit flaws in early operating systems and crash other users. After being banned by various service providers in the late 90s, he shifted interests abruptly to more affable projects, such as games and graphics which he continued through high school.

At Cornell University, Michael's undergraduate experience was haphazard. He was bored in early Computer Science courses and eventually opted for Electrical and Computer Engineering. His junior year, he accepted a nine month engineering co-op position with General Electric where he worked on data servers and software, and also coached little league baseball. In his senior year, he gave Computer Science another try, taking elective courses in machine learning and artificial intelligence. In particular, a course on evolutionary computation (using simulated evolution to explore open-ended intractable problems) triggered a lasting curiosity. His instructor,

Professor David Delchamps, later introduced Michael to Professor Hod Lipson. Shortly after, Michael graduated with a Bachelor of Engineering degree in Electrical and Computer Engineering in 2005, followed by a Master of Engineering degree in Computer Science in 2006. That same year, Michael was accepted into the Ph.D. program and commenced the work presented here.

*For Mary Westlund*

## ACKNOWLEDGEMENTS

I'd like to thank my advisor Hod Lipson for giving me direction and the capability to explore many fruitful and challenging topics. I'd also like to thank my research group, the Cornell Computational Synthesis Lab, for creative discussions on projects, and my committee advisors Professor Steven Strogatz and Professor Stephen Ellner for their valuable academic and professional discussions and advice.

I was fortunate to be supported by two graduate fellowships during my Ph.D.: the Integrative Graduate Education and Research Traineeship in 2007, and the National Science Foundation Graduate Research Fellowship in 2008.

More personally, I'd like to thank several people who helped me tremendously during my Ph.D. Thanks to Leticia Rojas for her unwavering moral support and inspiration to enjoy life. Thanks to Evan Malone for his help and support on everything from discussing physics to building custom server racks. Thanks to Michael Tolley, Jonathan Hiller, and Daniel Cohen for their friendship and support inside and outside of the lab. Thanks to Jonas Neubert, John Amend, Robert MacCurdy, and Jeffrey Lipton for many needed distractions and discussions. Thanks to Stephane Constantin, Nicolas Lassabe, and Simon Fivat for their friendship and help building dozens of servers. Thanks to Brian Herold for moral support and first introducing me to programming. Thanks to Aaron Lenfesty and Daniel Ly for their technical advice and coding discussions. And finally, thanks to my parents, Mary Westlund and Douglas Schmidt, for their love and support, and providing me freedom to pursue many opportunities throughout life.



## TABLE OF CONTENTS

BIOGRAPHICAL SKETCH.....	iii
DEDICATION .....	v
ACKNOWLEDGEMENTS .....	vi
TABLE OF CONTENTS .....	vii
LIST OF FIGURES .....	viii
LIST OF TABLES .....	xxiv
PREFACE.....	xxvi
SECTION I – INTRODUCTION.....	1
CHAPTER 1. GOALS .....	1
CHAPTER 2. MOTIVATION.....	1
CHAPTER 3. BACKGROUND.....	3
SECTION II – SEARCH METHODS .....	7
CHAPTER 4. FITNESS PREDICTION.....	7
CHAPTER 5. RANK PREDICTION .....	48
CHAPTER 6. META-OBJECTIVES IN EVOLUTIONARY SEARCH .....	63
CHAPTER 7. PRIOR MODELS AND SEEDING .....	79
CHAPTER 8. IDENTIFYING A DOMAIN ALPHABET .....	98
SECTION II – MODEL REPRESENTATIONS .....	117
CHAPTER 9. DYNAMICAL SYSTEMS.....	117
CHAPTER 10. IMPLICIT EQUATIONS .....	126
CHAPTER 11. NATURAL LAWS.....	142
CHAPTER 12. SYMBOLIC NOISE SOURCE MODELS .....	173
CHAPTER 13. STOCHASTIC REACTION MODELS.....	188
CHAPTER 14. TREE AND GRAPH ENCODINGS.....	204
SECTION III – INTERPRETING RESULTS .....	218
CHAPTER 15. PARAMETER MAPPING.....	218
CHAPTER 16. PARAMETER MODELS .....	248
SECTION IV – APPLICATIONS.....	251
CHAPTER 17. METABOLIC NETWORKS.....	251
CHAPTER 18. INSECT WING BUILDING BLOCK ANALYSIS.....	294
CHAPTER 19. USER PREFERENCE MODELING.....	308
CHAPTER 20. PUBLIC GOODS GAMES .....	332
CHAPTER 21. OPTICAL FILTERS .....	345
CONTRIBUTIONS .....	360
REFERENCES .....	366

## LIST OF FIGURES

Figure 4.1. High-level overview of the coevolution of solutions and fitness predictors algorithm.....	17
Figure 4.2. Pseudocode for the two threads in the algorithm that coevolve solutions and predictors. Trainers are chosen periodically in the predictor thread. ....	20
Figure 4.3. Pseudocode for pruning inactive expressions in randomly generated test problems to improve the complexity estimate for problem difficulty.....	22
Figure 4.4. The expected point evaluations before convergence versus the number of samples in the fitness predictor. Error bars show the standard deviation. ....	26
Figure 4.5. Histogram of training samples selected by the best fitness predictor during evolution to convergence of $f(x)=e^{ x }\sin(x)$ . Some samples are selected significantly more often than others. ....	27
Figure 4.6. The expected number of point evaluations before convergence versus the effort (percent of point evaluations) while training the fitness predictors averaged over 50 trials. Error bars show the standard error. ....	29
Figure 4.7. The training data of the three target functions experimented on. The horizontal axis shows the input values $x$ . The vertical axis shows the output training value $f(x)$ .....	31
Figure 4.8. The test set fitness during evolution for target functions $f_1(x)$ , $f_2(x)$ , and $f_3(x)$ respectively. Results are averaged over 50 trials. Error bars show the standard error. ....	32
Figure 4.9. Test set fitness versus evaluations averaged over 100 test runs for $f_2(x)$ . Error bars show standard error. ....	36
Figure 4.10. The Chi-square p-values for significance of convergence versus complexity between the coevolution algorithm and each compared algorithm. ....	39
Figure 4.11. The percent of successful convergence after 10 million point evaluations versus the target function complexity (the number of nodes in the binary expression tree).....	40
Figure 4.12. Improvement factor in convergence of coevolution over the other algorithms verses complexity for random target functions. ....	42
Figure 4.13. Fitness and percent of runs converged versus generations throughout evolution on the function $f_2(x)$ . Error bars show the standard error. Note that exact evaluations are performing significantly more computational effort per generation. ..	42

Figure 4.14. The size of the best solution during evolution of $f_2(x)$ averaged over 100 test runs. Error bars show the standard error. ....	44
Figure 4.15. The bloat of final converged solutions averaged over 500 randomly generated target functions. Error bars show the standard error. ....	45
Figure 5.1. The generation of random test problems for symbolic regression. We start by picking a random number of inputs, between one and ten. We then generate a random equation using these inputs and simplify the equation before measuring its complexity (the number of nodes in the binary tree). We then generate a random training data set by sampling the input variables around the origin and evaluating the target equation on these data points. We then generate a validation data set in a similar fashion, but with a wider range around the origin to test if the solutions extrapolate to the exact solution. ....	55
Figure 5.2. The fitness and convergence rate to the exact solution of each algorithm versus the total computational effort of each trial. The fitness (left) is the normalized mean absolute error on the validation data set. Convergence to the exact solution (right) represents the percent of the trials that identify solutions that have less than epsilon error on the validation data set. Error bars indicate the standard error. The performance of the algorithm without using prediction at all is several order of magnitude higher in computational effort and is not shown. ....	58
Figure 5.3. The computational effort required when the exact solution was found versus the target equation complexity (left) and the number of variables in the dataset (right). Each algorithm found the exact solution with different frequencies; these plots show the computation effort for when the algorithms did find the exact solution. The error bars indicate the standard error. ....	59
Figure 5.4. The mean solution bloat of the best solution versus the computational effort. Solution bloat is defined as the binary tree size of the best individual in the population minus the size of the target solution. Error bars indicate the standard error. ....	60
Figure 6.1. The novelty objective of a solution. Here, the novelty of equation #4 is equal to the maximum correlation of its residual errors with its two nearest neighbors in terms of fitness. ....	67
Figure 6.2. The Age-Fitness Pareto Population algorithm (A) considers a single population of individuals moving in a two-dimensional Age-Fitness Pareto space. Individuals are selected for if they simultaneous have higher fitness values and lower age than other individuals. Ages increase every generation, or are inherited during crossover, and new random individuals are added with zero age. In the Age-Layered Population Structure (ALPS) algorithm, there are several layers of populations for each age group. New individuals are injected to the youngest population, and individuals migrate to older populations as their age increases. ....	69

Figure 6.3. The fitness and convergence rate to the exact solution of the compared algorithms versus the total computational effort of the evolutionary search. The fitness is plotted (left) is the normalized mean absolute error on the validation data set. Fitness is normalized by the standard deviation of the output values. Convergence to the exact solution (right) is percent of the trials which reach epsilon error on the validation data set. The error bars indicate the standard error..... 72

Figure 6.4. Solution bloat over the course of the evolutionary search. Solution bloat is defined as the binary tree size of the best individual in the population minus the binary tree size of the target solution. The error bars indicate the standard error. .... 73

Figure 6.5. The performance of each combination of the multiple secondary objectives on random symbolic regression problems. Pane (A) shows the mean absolute error on the test data set of the best solution found by each algorithm. Pane (B) shows the convergence rate, the percent of times each algorithm identified the exact solution. Pane (C) shows the percentage of the Pareto space, defined by solution error and solution complexity (the two metrics of interest in the Symbolic Regression), that each algorithm explored..... 75

Figure 6.6. The convergence (percent of problems where each method identified the exact solution) versus the problem complexity. These results are split into three panes to make the differences more easily identifiable. Pane (A) shows the results for combinations of two objectives plus the single error objective. Pane (B) shows the results for three objectives plus the best 2 objective method and error objective. Pane (C) shows the best of the previous panes with the 4 objective method..... 77

Figure 7.1. Example seed equations for each method (left) and an example randomly generated target equation plotted next to the automatically generated approximate equation (right). .... 82

Figure 7.2. The expected time for the evolutionary search to converge to the exact target equation for each seeding method measured in function evaluations (runs that did not converge omitted). Error bars show the standard error. .... 89

Figure 7.3. The mean fitness (top) and convergence rate (bottom) for each method measured over each evolutionary trial. Error bars show the standard error. .... 90

Figure 7.4. The logistic trends of each seeding method in convergence rate versus target equation complexity (top), and linear trends in convergence versus the error of the approximate seed equation from the target equation (bottom). Error bars show the range based on the standard errors of the trend fit parameters..... 92

Figure 7.5. The solution bloat of the top ranked solution over the evolutionary runs. Bloat is measured as the top ranked equation’s complexity minus the target equation complexity. Error bars show the standard error. .... 95

Figure 8.1. We distill the common mathematical language needed to describe a group

of systems using symbolic regression and analysis of their model accuracy/complexity Pareto fronts. We generate experimental data from several related systems such as spring and mass mechanical systems (left). We then use symbolic regression to find several accurate models at varying complexity of equations (middle). Finally, we discompose models on these fronts to individual terms and building blocks. The most frequently used terms and building blocks form an emergent alphabet for describing models of this group of systems (right). ..... 99

Figure 8.2. Example equation (a), its binary parse tree (b), and all possible building blocks of the equation (c). Building blocks are common sub-expressions or internal components of a system that simplify building a full mathematical model. .... 102

Figure 8.3. Summary of the mechanical systems, the collected data of their dynamics, and the resulting models found using symbolic regression on the equation accuracy and complexity Pareto front. Each system was simulated numerically. The symbolic regression algorithm generates a small set of equations for each system. This set is a Pareto front, showing the most accurate equation found for different sizes (complexities) of equations. These equations are used to distill a common mathematical alphabet of building blocks for modeling mass, spring, and pendulum mechanical devices. .... 109

Figure 8.4. The building blocks found for the domain alphabet based on the harmonic oscillator, simple pendulum, and 2D spring pendulum Pareto front models. The most frequent and complex building blocks correspond to the kinetic energy terms for moving masses and potential energy terms for springs and pendula. Building blocks with zero frequency on the Pareto fronts of other systems are omitting and not included in the alphabet. .... 111

Figure 8.5. The impact of using a domain alphabet obtained from simple systems, the harmonic oscillator and simple pendulum, to find the model of a more complex system, the 2D spring pendulum. The alphabet in (top) shows the common building blocks found from the Pareto analysis of only the harmonic oscillator and simple pendulum systems. Allowing symbolic regression to use these terms substantially accelerates the modeling of the more complex 2D spring pendulum system (bottom). Error bars show the first standard error about the mean over ten independent trials. 114

Figure 10.1. Many datasets exist that do not have explicit dependent variables, such as an elliptic curve shown here. Instead, this type of data must be modeled with an implicit equation. We explore using symbolic regression to infer these types of models. .... 128

Figure 10.2. Implicit derivatives can be estimated from unordered, or shuffled data, non-parametrically by fitting a hyperplane or higher-order surface to neighboring points. After fitting the neighboring points, simply take any of the implicit derivatives of the locally fit surface. .... 134

Figure 10.3. Data sampled from six target implicit equation systems. Data is collected uniformly for the geometric systems. In the dynamical systems, the data is a single simulated trajectory from a random initial condition. .... 136

Figure 10.4. Fitness of the symbolic regression algorithm using the implicit derivatives fitness for each of the six systems. Results are the top ranked solution versus time, averaged over 20 independent trials. Error bars indicate the first standard error. .... 138

Figure 10.5. The fitness and equation complexity Pareto fronts found for each of the six systems. The exact solutions are the simplest equations to reach near perfect fitness. More complex solutions show elaborations on the exact solution, improving fitness only marginally. .... 140

Figure 11.1. Mining physical systems: We captured the angles and angular velocities of a chaotic double-pendulum (A) over time, using motion tracking (B), then automatically searched for equations that describe a single natural law relating these variables. Without any prior knowledge about physics or geometry, the algorithm found the conservation law (C), which turns out to be the double-pendulum's Hamiltonian. Actual pendulum, data and result shown. .... 143

Figure 11.2. The computational approach for detecting conservation laws from experimentally collected data. (A) First, calculate partial derivatives between variables from the data, then search for equations that may describe a physical invariance. To measure how well an equation describes an invariance, derive the same partial derivatives symbolically to compare with the data. Finally, return the most parsimonious equations for the hypothesized physical laws. (B) The representation of a symbolic equation in computer memory is a list of successive mathematical operations. (C) This list representation corresponds to a graph, where nodes represent mathematical building blocks and leaves represent parameters and system variables. Both (B) and (C) correspond to the equation  $f(\theta, \omega) = 17.719 - 4.771\omega^2 + 4.714\cos(\theta) - \omega^2\cos(\theta)$ . To search for conservation equations, the algorithm mutates and recombines these structures to search the space of equations. .... 146

Figure 11.3. Summary of laws inferred from experimental data collected from physical systems. Depending on the types of variables provided to the algorithm, it detects different types of laws. Given solely position information, the algorithm detects position manifolds; given velocities the algorithm detects energy laws; given accelerations, it detects equations of motion and sum of forces laws. These laws contain bulk parameters. .... 149

Figure 11.4. Parsimony vs. accuracy, and performance. (A) The Pareto front (solid black curve) for physical laws of the double-pendulum and the frequency of sampling during the invariant equation search (grayscale). The Pareto front shows the trade-off between equation complexity (or parsimony) and ability to model a predictive invariance. At a critical complexity of  $\sim 32$ , there is a strong point of inflection. The

equation at the inflection corresponds to the exact energy conservation law of the double-pendulum, highlighted. A second momentum conservation law encountered is also highlighted. (B) The computation time required to detect different physical laws for several systems. The computation time increases with the dimensionality, equation complexity, and noise. A notable exception is the bootstrapped double pendulum, where reuse of terms from simpler systems helped reduce computational cost by almost an order of magnitude, suggesting a mechanism for scaling higher complexities..... 152

Figure 11.5. Ancestor trajectories in equation space while searching for the equation of an ellipse. Dots indicate crossover and mutation events while lines represent parameter tuning over time. (A) Several initially random equations with varying predictive ability evolve independently before coalescing toward the exact solution over the running time of the algorithm. (B) The ancestors also vary in equation complexity – measured as the number of nodes in their expression trees. Initial equations tend to have higher complexity, but simplify over time toward the exact solution. (C) The same trajectories plotted over predictive ability and complexity shows the ancestor trajectories converge toward a simple and high predictive ability neighborhood before finding the correct equation structure whose parameters can be tuned to the exact solution. .... 160

Figure 11.6. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Figure 11.5. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation. .... 161

Figure 11.7. Two equivalent representations of an example equation  $f(\theta, \omega) = 17.719 - 4.771 \cdot \omega^2 + 4.714 \cdot \cos \theta - \omega^2 \cdot \cos \theta$ . (A) The algorithm stores and evolves equations represented by a list of floating point operators over a system's variables. Each operation can load a variable, load a parameter, or perform a mathematical operation on any previous operation. Unused lines have been omitted for clarity. (B) The raw list can be interpreted more intuitively by an acyclic graph where several sub-trees are reused by multiple terms. Both (A) and (B) represent the same equation. .... 163

Figure 11.8. The accuracy/complexity Pareto front of the double pendulum. The Pareto front shows the tradeoff between equation complexity and its ability to derive accurate partial derivative. At some minimum complexity (32 nodes), predictive accuracy jumps rapidly. Equations almost twice as complex improve the accuracy only marginally. These high complexity equations tend to contain the simpler exact equation, but add many smaller terms to compensate noise. The parsimonious and accurate equation at the inflection is the Hamiltonian and Lagrangian of the double pendulum. .... 168

Figure 11.9. The mean predictive ability on a withheld test set of the best equations detected versus the amount of normally distributed noise in the data set for the simulated double linear oscillator. Error bars show the standard error. The percent

noise is the ratio of the standard deviation of the noise and the standard deviation of the original signal. .... 169

Figure 12.1. Three basic examples where a stochastic element hides or distorts analytical features of the system to different extents. Blue dots show the observed system output, the red line shows the expectation of the output, and the green line show the target analytical model with stochastic elements removed. .... 177

Figure 12.2. Pseudocode for evaluating a model with stochastic noise sources to estimate the noise envelope or distribution (top), and pseudocode for calculating the resulting fitness metric for the candidate model (bottom). .... 179

Figure 12.3. An example binary expression tree (a) for the function  $f(x) = e^x \sin(x)$ , and a similar tree modeling a stochastic element (b) for the function  $f(x) = e^x \sin(x + R())$ . .... 180

Figure 12.4. The fitness objective for explaining training data with a with model that has stochastic elements and output distribution. If a training point falls inside the model distribution, the objective is to minimize the height of the distribution. If the point falls outside, the objective is to minimize the distance of the point to the distribution. .... 182

Figure 12.5. The best model found at three points during regression of  $f(x) = 10 \sin(x + R)$ . The green points show the training data, the grey area shows the model's distribution, and the blue line shows the analytical model with stochastic elements removed. .... 185

Figure 12.6. The best model found at three points during regression of  $f(x) = x^2 \sin(x + R)$ . The green points indicate the training data, the grey area indicates the model's distribution, and the blue line indicates the analytical model with stochastic elements removed. .... 185

Figure 12.7. The best model found at three points during regression of  $f(x) = (x + R) - 1.5 x^3$ . The green points are the training data, the grey area is the model's distribution, and the blue line is the analytical model with stochastic elements removed. .... 185

Figure 13.1. Overview of the modeling problem. A stochastic system evolves an exact behavior over time shown in blue. Periodically, the state of system can be measured (shown in red dots), a sample of the exact time evolution of the system. The task is to infer a maximum likelihood stochastic model (right) for this system from these periodic measurements. Actual data and solution shown. .... 189

Figure 13.2. The encoding of a solution representing a stochastic model of discrete reactions. A series of chemical reactions (top) are represented by corresponding integer coefficients and real valued rate constants for each reaction (bottom). .... 193

Figure 13.3. Comparing a candidate model with the experimental data. The left pane



shows the hypothetical exact behavior of a system in blue, and two known measurements of the system at red dots. The candidate model is simulated multiple times, starting from the first measurement for  $t$  seconds, in order to estimate a probability distribution of the model (right). The state of the second measurement is then compared with this distribution to evaluate the quality of the model to reproduce the measurement. .... 196

Figure 13.4. The search performance of the three compared fitness metrics. The top panes show performance when data points appear in rapid succession with short gaps in time. The bottom panes show performance when there are long gaps of time between data points. The left panes show the likelihood score of the best model during the search. The right panes show the percent of runs that identified the exact solution for the amount of computational effort. Error bars indicate the standard error. .... 198

Figure 13.5. The relationships between the distance metric of a model and its corresponding likelihood given the experimental data. Each point in the plot is a random candidate model during the likelihood search. .... 200

Figure 13.6. Traits of the best model over time during the evolutionary search. The top left plot shows the genotypic age of the best solution (the number of generations any part of the solution existed in the population). The top right shows the novelty of the best solution (how different it is from the rest of the population). The bottom pane shows the bloat of the best solution (ratio its complexity with the target solution complexity). Error bars indicate the standard error. .... 201

Figure 14.1. Example expressions of  $f(x) = (x + 1)^4$  in the tree encoding (a) and graph encoding (b). The graph encoding reuses redundant sub-expressions but is more susceptible to deleterious variation. .... 207

Figure 14.2. Bloat of converged solutions for 1-variable functions (a), and 8-variable functions (b). Each point is averaged over 50 randomly generated target functions. Error bars show the standard error. .... 212

Figure 14.3. Test set convergence versus target function complexity for 1-variable functions (a), and 8-variable functions (b). Each point is corresponds to 50 randomly generated target functions. .... 213

Figure 14.4. The number of point evaluations before convergence on the training set versus the target function complexity for 1-variable functions (a), and 8-variable functions (b). Points are averaged over 50 randomly generated target functions. Error bars show the standard error. .... 214

Figure 14.5. The rate of beneficial crossovers versus target function complexity for 1-variable functions (a), and 8-variable functions (b). Results are averaged over 50 random test problems. Error bars show the standard error. .... 215

Figure 14.6. The point evaluations per second versus the function complexity. .... 216

Figure 15.1. Manually-derived versus automatically-generated biological models and the mapping challenge. Most biological models are derived by hand using expert knowledge of the system, related systems, and qualitative understandings of the underlying biology (left). When large amounts of experimental data are available, empirical models can be inferred automatically by a computational search for the most parsimonious model that accurately predicts the dynamics (right). The automatically-generated model potentially provides new insight into the system but does not have any accompanying explanation. Our solution to this problem is to additionally learn a mapping from the known biological model to the automatically-generated model, identifying which understood parameters collapse to simpler explanations in the automatically-generated solution. Actual models and data shown.  $K$  and  $S$  represent the protein concentration levels of *ComK* and *ComS*, respectively.  $\alpha$ , and  $\beta$  terms correspond to the basal and maximum rates of protein expression, respectively.  $\lambda$  denotes the linear and  $\delta$  the enzymatic degradation rates of *ComK* and *ComS*. The meanings of the parameters on the right are unknown. .... 221

Figure 15.2. Transient and oscillatory dynamics of competence events in single *B. subtilis* cells. Filmstrips in panels A and B show overlays of phase contrast and two-color fluorescence images. Blue and orange colors depict the reporter for competence *PcomG* and negative feedback loop component  $P_{comS}$ , respectively. Panel A shows a single wild type cell that differentiates into the competence state and then exits (indicated in blue). Panel B, shows cells containing a modified competence circuit (for details see text and SOM) that give rise to oscillations in competence where cells undergo consecutive events. Panels C and D depict time traces of promoter activity obtained from quantitative image analysis of fluorescent reporters during the competence events shown in panels A and B respectively. Blue and orange colors utilized in the graphics are consistent with the colors depicted in the filmstrips and time traces, where blue indicates competence and orange the activity of the negative feedback loop necessary for exit from competence. .... 224

Figure 15.3. The automated modeling method attempts to model multiple cells with a single equation, and then identify a nonlinear mapping to a previous understood model. These equations contain symbolic parameters which vary for each cell, rather than constant coefficients. The algorithm searches for the most parsimonious equation which accurately predicts the dynamics observed in the experimental data using an evolutionary search. We then attempt to identify a mapping of this model to the currently understood system model by varying parameters of the manually-derived model, simulating it numerically to generate new data, and then fitting the automatically-generated model to the generated data. We then search for a nonlinear relationship between the parameters of the two models. .... 226

Figure 15.4. The automatically-generated conserved quantity (A) maps onto a small set of parameters in the manually-derived model (B) which correspond to the degradation of *ComK* and production of *ComS* (C). When evaluating the conserved quantity on data collected from two different types of *B. subtilis* strains (D), a sort duration strain (black) and a longer duration strain (red), the magnitude of the

conserved value separates into two different groups (E), suggesting the conserved quantity is tied to the duration of competence events. .... 229

Figure 15.5. The mapping between the manually-derived model and the automatically-generated dynamical model connects the simpler data-driven model with the current biological understanding. The bipartite graph (B) shows the linear correlation strengths between model parameters – automatically-generated model parameters are on the left side, manually-derived model parameters are on the right side. The nonlinear mapping (C) shows that multiple parameters of the manually-derived model collapse to those in the simpler automatically-generated model. The parameter plots (A) show that the mapping is in strong agreement with the automatically-generated model over a wide range of parameter values. .... 232

Figure 15.6. Collected data and the fit of the automatically-generated dynamical model. *ComK* florescence (AFU) is shown in blue dots, *ComS* florescence (AFU) is shown in red dots, and the automatically-generated model is shown in black for each. The automatically-generated model was found using data from the top four rows. The bottom row shows that the model generalized to other behaviors such as oscillating competence events. .... 238

Figure 15.7. The parameter mapping relating the parameters of the expert biological model and the automatically identified dynamical model. The left plots show the predicted parameter value in the automatically-generated model based on the parameters of the expert model versus the actual best fit parameter of the automatically-generated model. The parameter equations found are shown to the right. The percent shown for each term indicates the percent of the variance explained by each term. .... 240

Figure 15.8. The parameter mapping relating the parameters of the expert biological model to the automatically-inferred conserved quantity. The left plots show the predicted parameter value in the conserved quantity of the mapping versus the actual best fit parameter of the conserved value. The parameter equations found are shown to the right. .... 242

Figure 15.9. The clusters of coefficient values of the unknown conserved quantity equation colored by the *B. subtilis* strain. Each plot shows a projection onto a different pair of coefficients. .... 244

Figure 15.10. Verifying the perturbations of the models with the physical changes in the wild (black) and mutated (red) strains. Pertubing only the parameters that correspond to production of *ComS* in the simulated model produces similar changes to those seen in experiment. .... 246

Figure 17.1 Automated analytical modeling: Noisy time series data reflecting anaerobic metabolism concentrations over time are automatically translated into a set of coupled analytical differential equations without prior knowledge of the system

(actual data and equations). ..... 252

Figure 17.2. Analytical model representations for NADH in the cell glycolysis model - a tree encoding (left pane) and a graph encoding (right pane). Both panes encode the same equation, but while the tree encoding is simpler to manipulate algorithmically (*e.g.*, alter subexpressions), it requires redundant subtrees and is prone to produce large equations that may not accurately represent the biological system. The graph encoding couples subtrees, thereby biasing equations to preserve simpler shared expressions. .... 258

Figure 17.3. The pareto front of model accuracy versus its simplicity. There is an inherent trade-off between complexity and accuracy to the training data. Many complex functions have very high accuracy, however the exact solution lies at the sharp inflection near 28 nodes, balancing high accuracy and simplicity. .... 262

Figure 17.4 The coevolution of models through symbolic regression and fitness prediction, and experiments by the estimation-exploration algorithm. Candidate solutions compete to explain current experimental data, and experimental initial conditions compete to maximize disagreement in the predictions of the various solutions. This process of synthesizing coherent models and controversial experiments continues until a single dominant solution emerges. .... 264

Figure 17.5. The residual squared-error after Loess smoothing versus the magnitude of the noise and the density of features relative to the noise frequency (sample rate) for a sine-wave signal and its numerical derivative. The signal error is most sensitive to the noise magnitude but more robust to the number of features. In contrast, the error on the numerical derivative has much higher sensitivity to the number of features. The state of the art of what the symbolic regression algorithm can handle with Loess smoothing is roughly the medium-blue to dark-blue regions. .... 266

Figure 17.6. Reaction networks for anaerobic metabolism in a yeast cell. Left: The exact model includes membrane transport of glucose and pyruvate/acetaldehyde. Reactions in red involve ATP production/usage, and reactions in blue involve redox species production/usage. Middle: The impaired model does not produce either glycerol or ethanol. Right: The overspecified model has an additional sink for pyruvate/acetaldehyde ( $S_4$ ). .... 275

Figure 17.7. The fit to the data of the highest ranked solution during regression for each glycolysis variable. The blue series show the correlation coefficient to the training data, and the red to the test data. The training data contain 10% noise while the test data have none. The test data contain a larger range of allowed state variables (*i.e.*, sampled with weaker constraints) to measure whether the model can extrapolate and predict new behavior. .... 278

Figure 17.8. The exact black box model and inferred model integrated over time. The inferred model shown in Table 17.5 differs from the exact model by a slight mass

imbalance. Integrated over 10 minutes, the inferred model captures the same behavior. While small differences in derivative values tend to accumulate during integration, the inferred model captures the integrated behavior remarkably well. The inferred model predicts early behavior accurately and exhibits the same qualitative dynamics later in time, differing only slightly in the phase. .... 280

Figure 17.9 The glycolysis system near the stable limit cycle in the course of a single experiment, with colors representing frequency with which the fitness predictor examines each point within a single time-series. .... 281

Figure 17.10. The initial condition experiments (red) chosen by the algorithm to differentiate solutions in comparison to a random distribution of initial conditions (blue). The algorithm tends to focus on nonlinear states away from the limit cycle (dashed black line) within the experimental constraints imposed upon the estimation-exploration algorithm. .... 282

Figure 17.11. (A) The rate of successful inference of the exact differential equation for each state-variable versus the observation noise in the system after one hour of regression. The convergence rate is calculated from ten independent trials on each equation at each noise level. (B) The rate of successful inference of the exact differential equation for all variables versus the total amount of data given to the system. The error bars indicate the standard deviation in convergence among the seven variables. .... 284

Figure 17.12. Performance comparison between symbolic regression, nonlinear regression, and neural network regression. Results are averaged over 100 trials – error bars represent the standard error. Training data performance (top pane) shows that all algorithms accurately explain the training data. The negative slope of the correlations when the results from the training regression are applied to the test data indicates varying degrees of overfitting. Note that symbolic regression uses more point evaluations in the same amount of running time because it is a parallel search, whereas nonlinear regression and neural network back-propagation use serial updates. .... 287

Figure 17.13. Correlations of the various regressions averaged over 100 trials on equation  $S_4$  – error bars represent the standard error. (A) The correlations between the training data and each initial model before the model is regressed to the training data by the corresponding algorithm. Symbolic regression and neural network regression must model the system from scratch and initially have zero correlation. The impaired and overspecified models are close approximations to the exact model and therefore have positive correlations. (B) The mean correlation of the best solution from ten runs of each algorithm to the training data. The training data contain 10% random noise, which results in slight variances – most notably in the neural networks. The best solution from each algorithm correlates well to the training data with low standard error. (C) The mean correlation of each method to the test data. The assumed structures of the impaired and overspecified models limit their ability to model a wider phase domain. The neural network appears limited by noise in the system, but does

achieve a higher correlation on average with the test set than do the impaired and overspecified models. .... 290

Figure 17.14. Performance comparison of symbolic regression when correcting a hypothesized model. Results are averaged over 100 trials – error bars represent the standard error. The blue curves represent the performance of the algorithm to the  $S_4$  equation without any prior model. For the other two pairs of curves, the symbolic regression algorithm was seeded with an incorrect hypothesized model (black = impaired, red = overspecified) and the algorithm had to modify the seeded model to fit the original training data. The graph shows the performance for both the training data used for the regression (top pane), and the test data (bottom pane) used to evaluate the training regression. .... 291

Figure 18.1. The tracked position of the fly (top pane) and the corresponding angles of the right wing (bottom pane). This data was recorded over approximately 34 flapping periods during 140 milliseconds of flight. .... 296

Figure 18.2. The three dimensional plot (left pane) of the right wing angles ( $\theta_1$ ,  $\theta_2$ , and  $\theta_3$ ). There is slight variation among the periods but overall they line up neatly. After chopping up the periods, there is covariance between different peaks of each angle (right pane). .... 297

Figure 18.3. Phase plots of the three angles of the right wing ( $d\theta_i/dt$  vs.  $\theta_i$ ), in order ( $\theta_1$ ,  $\theta_2$ , and  $\theta_3$ ). The first angle appears to be a simple harmonic oscillator, whereas the other two angles show more complex sub-cycles, likely containing higher order building blocks in their physical model. .... 298

Figure 18.4. Functional linear models based on the period number explains much of the variation between periods. The linear coefficients (left), the fit and description of the drift (middle), and  $R^2$  scores (right). .... 299

Figure 18.5. The registration method slices the data into each periods, scaling length of each slice to have the same period (left pane). The method optimizes the positions of the slices in order to maximize the correlation among all the periods (right pane). .... 300

Figure 18.6. The registered data (left pane) and the shifts in periods after optimizing the slice positions (right pane). The periods over time drift slowly, correlating with the slight drift in the position of the fly in Figure 18.1. .... 301

Figure 18.7. An equation modeling  $D^2\theta_2(t)$  and its individual building blocks. This equation was generated by an algorithm, so we are interested in testing whether its building blocks also show to be useful individually using the function linear model procedure. .... 303

Figure 18.8. The cross-validation error of the functional linear model using various building blocks for  $D^2\theta_2(t)$ , (red lines) shown in Fig. 8 and the null functional linear

model (dashed black lines) versus the smoothing penalty  $\lambda$ . In each case the null model is dominated by the building block model for all coefficient smoothing penalties.... 304

Figure 18.9. Cross-validation error of several poor building blocks (solid red) and the null model (dashed black). The poor building blocks fail to either achieve lower minimum error or tolerate higher smoothing penalties. The poor building blocks, other than  $t$ , were also building blocks generated by the modeling algorithm. .... 305

Figure 19.1. Example comparison between two pen drawings. .... 312

Figure 19.2. The user interface presented to the user. .... 313

Figure 19.3. Example comparison relational graph of six individuals. .... 314

Figure 19.4. Example Pareto front plot of eight potential comparison pairs. .... 315

Figure 19.5. The basic structure of an individual comparator user model. .... 316

Figure 19.6. The structure of a neural network user model. .... 317

Figure 19.7. The comparator model based interactive evolution algorithm basic outline. .... 319

Figure 19.8. The prompts given to the user and the resulting top three guesses over six iterations. .... 324

Figure 19.9. The number of user prompts expected between the compared algorithms to find a square shape. .... 325

Figure 19.10. The prompts given to the user and the resulting top three guesses over seven iterations. .... 327

Figure 19.11. The number of user prompts expected between the compared algorithms to find the target star shape. .... 328

Figure 19.12. The clock time fitness landscapes calculated over six user prompts. .. 331

Figure 20.1. The Logistic Function squashes all inputs to the range from zero to one. This function is used in the model structures because we are modeling a fraction value (e.g. the fraction of money invested in the public good). The input to the logistic function then has an interpretation of a strength toward zero (large negative values) or one (large positive values). .... 335

Figure 20.2. The model obtained for an individual's contribution to the public good ( $x$ ) in the normal Public Goods Game. The left pane shows the correlation of the model predictions with the data. The right pane shows the predictions of the model (the 3D surface) next to the experimental data (the blue dots). The model suggests that a player

is less likely to contribute if they have high cumulative earnings relative to the group, however they are more likely to contribute if others in the group contribute. The fitted parameters are  $\alpha = 3.51853$ ,  $\beta = 6.21075$ ,  $\gamma = 5.08922$ . ..... 336

Figure 20.3. The fitness landscape for an individual against the field assuming that the group behaves according to the model. The individual's fitness improves for increasing group contribution ( $\text{group}(x)$ ) and decreasing individual contribution ( $x$ ). Thus, the model predicts that players will contribute less and less, tending toward zero contribution..... 338

Figure 20.4. The model obtained for an individual's contribution to the public good ( $x$ ) in the tug-of-war version of the game. The left pane shows the correlation of the model predictions with the data. The right pane shows the predictions of the model (the 3D surface), next to the experimental data (the blue dots). This model suggests if the player is doing well in cumulative earnings and kept and/or competed an amount previously, she/he is less likely to contribute. The fitted parameters shown are  $\alpha = 1.62552$ ,  $\beta = 4.61882$ ,  $\gamma = 1.37634$ . ..... 339

Figure 20.5. The model obtained for an individual's fraction kept ( $y$ ) in the tug-of-war version of the game. The right pane shows the correlation of the model with the data. The model predictions (the 3D surface) is plotted next to the experimental data points (blue dots). The model suggests that if the group is contributing, the player is less likely to keep. However, if the player has been successful in total earnings and kept in the previous round, she/he is more likely to continue keeping. The fitted model parameters shown are  $\alpha = 0.0319059$ ,  $\beta = 4.52439$ ,  $\gamma = 1.90551$ . ..... 340

Figure 20.6. The model obtained for an individual's fraction invested in competition ( $z$ ) in the tug-of-war version of the game. The left pane shows the linear correlation of the model predictions with the data. The right pane shows the model predictions (the 3D surface) next to the experimental data. The model suggests that if the group is keeping and the player kept on the previous round, she/he is less likely to compete. Secondly, if the group is contributing and the player competed in the previous round, she/he is more likely to compete again. The fitted model parameters shown are  $\alpha = 0.81949$ ,  $\beta = 1.44882$ ,  $\gamma = 4.38114$ . ..... 342

Figure 20.7. The effective fitness landscape for an individual playing against the field, assuming the group plays according to the  $x_{n+1}$  and  $y_{n+1}$  models, and in the previous round the group played 50% in contribution ( $x_n$ ) and 50% in competition ( $z_n$ ). The optimal behavior for the individual in this circumstance is to play similarly: investing at least 50% in competition and the rest in contribution. The fitness surface predicted by the model looks similar for other group conditions, most with optima at  $z_n = 100\%$ . ..... 343

Figure 21.1. Ring-resonator device structure. Each component contributes to the final transmission. .... 346



Figure 21.2.. Example filter encoding with a maximum of three devices. Each device consists of seven independent parameters and a flag to include or omit the device from the final series. In this figure, device #2 is flagged as omitted. ....	351
Figure 21.3. Evolved low-pass filter. The target transmission is shown as a dotted line and the best evolved solution is shown in solid. This solution used three devices in its encoding, shown in the right pane. ....	352
Figure 21.4. Evolved box filter. The target is shown as a dotted line, the best evolved solution is shown in solid. This solution used five devices in its encoding, shown in the right pane. ....	353
Figure 21.5. Evolved band-pass filter. The target transmission is shown as a dotted line, the best solution is shown in solid. This solution used four devices in its encoding, shown in the right pane. ....	353
Figure 21.6. Evolved ramp filter. The target transmission is shown as a dotted line, the best evolved solution is shown in solid. This solution used two devices in its encoding, shown in the right pane. ....	353
Figure 21.7. A damaged five-device filter. <i>The <math>\kappa</math>, <math>\alpha</math>, <math>\Phi_0</math>, <math>R</math>, and <math>G</math> parameters are offset by 10% random manufacturing error. The qualitative box-filter transmission function has been restored however some precision is still lost. ....</i>	354
Figure 21.8. Recovering a damaged device – box, LPF, and ramp filters. The left pane shows the error of the best filter being evolved before fabrication. 10% random error is then added to all fixed parameters on all devices. The right plane shows the best filter being evolved to recover from manufactured errors. Errorbars show the standard error. ....	356
Figure 21.9. Inferring the physical parameters of a 2-device filter. The inferred model matches the hidden system to within very low error. Note that the order of devices and the gain levels of each individual device cannot be determined due to algebraic properties of multiplying the transmission of each device. The total gain is inferred correctly however ( $1.312*0.784 = 1.002*1.026 = 1.0028$ ). ....	357
Figure 21.10. Reverse-engineering random 4-device filters give precise transmission measurements (blue) and noisy transmission measurements (green). Error bars show the standard error. ....	358

## LIST OF TABLES

Table 4.1. Summary of the Compared Algorithms .....	30
Table 4.2. Performance comparison to published methods.....	35
Table 4.3. Example functions and complexities.....	37
Table 4.4. Chi-Square Significance of Convergence Rates Compared to the Coevolution Algorithm.....	38
Table 9.1. Fitness prediction algorithm parameters .....	119
Table 9.2. Inferring various physical and biological dynamical models.....	123
Table 10.1. A summary of direct methods and their difficulties .....	132
Table 11.1. The predictive ability and Pareto fronts of several synthetic manifolds and simulated dynamical systems. Error bars denote the standard error of predictive ability .....	165
Table 11.2. Summary of Detected Approximations with Missing Building Blocks..	171
Table 12.1. Summary of Experiment Setup .....	183
Table 14.1. Summary of Experiment Setup .....	209
Table 17.1. Raw encodings of glycolysis differential equations found. ....	260
Table 17.2. The chemical species in the model (NM, IM, and OS are the normal, impaired, and overspecified models, respectively). ....	268
Table 17.3. Description of the reaction fluxes and their kinetic coefficients.....	269
Table 17.4. Model variables, the allowed range of initial states for the training data set, and the standard deviation of the limit cycle used to compute the amount of added noise.....	271
Table 17.5. The differential equations describing glycolytic oscillation of the generating model (left pane) and the inferred model from the training data, which had 10% noise (right pane).....	277
Table 17.6. Seven snapshots of the best solution during regression of $S_1$ . The solution is plotted in red and the systems limit cycle is shown in blue.....	285
Table 17.7. The equations for $S_4$ (pyruvate and acetaldehyde pool) for the exact, impaired, and overspecified models shown in Figure 17.6. The exact values for the	

parameters are $k_3 = 16$ , $k_4 = 100$ , and $A_s P/V = 13$ . .....	288
Table 19.1. Neural Network Training Parameters.....	320
Table 19.2. Evolution Parameters. ....	321
Table 20.1. Model symbol definitions.....	333
Table 20.2. Model symbols for group averages. ....	334
Table 21.1. Range of parameters describing each doubled ring resonator. The parameters controlled externally by all-optical effects are $\phi_1$ and $\phi_2$ . ....	347
Table 21.2. The parameters for the ring device with transmission function shown in Figure 2 as the dotted line. We simulate manufacturing errors on this device by adding 10% random errors to all parameters. The damaged transmission function is shown in Figure 2 as a dashed line. ....	348

## PREFACE

The area of automated modeling contains many unexplored directions, and I had to work on a breadth of topics as they came up over the course of my Ph.D. While compiling this text, I had some initial concern that the length was too long. I thought carefully about excluding large portions. However, I decided to keep the text complete so that it could serve as a comprehensive record of my work. The final dissertation comprises all of my research related to automated modeling during my Ph.D. It includes both topics that I have published on and also smaller unpublished results. The chapters span a breadth of topics and several explore significant depth into various problems. Still, there are many remaining questions to be answered in this growing field. I hope this text may inspire others in future directions.

## SECTION I – INTRODUCTION

### CHAPTER 1. GOALS

The central goal of this work is to advance new technology to accelerate scientific discovery. In particular, this work focuses on Automated Modeling and Artificial Intelligence for analyzing experimental data observed in a physical system in order to hypothesize about its analytical rules and intrinsic relationships; ultimately helping to transform data into scientific knowledge.

Scientific discovery often progresses in stages, from making observations and performing experiments (data), to modeling and predicting the outcome of experiments (predictions), to identifying the symmetries and rules of the phenomenon (laws), and finally to developing theories and understanding (meaning). The work presented here explores computational methods to move from data to laws, leaving humans to take the last step.

This work addresses the task in four core areas: improving search methods and algorithm performance, improving model representations and expanding the types of solutions that can be modeled, interpreting results and connecting to them previous knowledge, and finally proving these techniques to realistic systems.

### CHAPTER 2. MOTIVATION

In 2006, Josh Bongard, Viktor Zykov, and Hod Lipson developed a continuous self-modeling robot (Bongard, Zykov et al. 2006) – a robot that could, using only raw data from its internal sensors, deduce its own configuration. For example, the robot could determine that it had four legs of specific lengths and orientations. Even after a leg of

the robot was broken off, the robot would refine its model and design a new gait to continue its locomotion.

This work begged the question: could a similar robot also model external phenomenon? This concept led Bongard and Lipson to pioneer new research in automated modeling of dynamical systems (Bongard and Lipson 2007), and formed a basis for future work in automated modeling of dynamical systems, including the work presented here.

Automated methods for generating, collecting and storing data from experiments have become increasingly precise and efficient over the past decade (Clery and Voss 2005; Szalay and Gray 2006). But the technology to make hypotheses or convert data into meaningful analytical relations hasn't kept pace. As a result, there is increasing interest in new forms of automated analysis, and automating tasks which traditionally required human labor and expertise.

Many methods already exist for modeling scientific data: from fixed-form parametric models derived from expert knowledge to statistical models aimed exclusively at prediction. However, there exist very few methods for creating human-understandable models of nonlinear systems from experimental data.

Recently, the ongoing research to address this problem has accumulated several different names, from “Machine Science” (Evans and Rzhetsky) or “Automation of Science” (Waltz and Buchanan 2009) to “The Robot Scientist” (King, Whelan et al. 2004). The actual machines comprising these systems remain less glamorous than their names imply (e.g. a rack of servers in a data center). But, there is increasing debate (Gianfelici 2010; Haufe, Elliott et al. 2010; Leonelli 2010) that our concept of science, and what it means to do science, may be changing (Mitchell 2009).

Of course not everyone agrees – Philip Anderson and Elihu Abrahams have claimed that there is “no mechanism by which [machines] could create a Kuhnian revolution and thereby establish new physical law” (Anderson and Abrahams 2009).

Regardless of the various predictions on the future, this is a critical question that must be answered. And, it is the motivating factor for the work presented here.

### CHAPTER 3. BACKGROUND

This section briefly describes essential information describing background concepts and previous research that are referenced in several chapters of the text. It covers evolutionary computation and symbolic regression. Individual chapters also contain their own specific background topics; the following topics are common to almost all chapters.

#### *Evolutionary Computation*

An evolutionary algorithm is an optimization algorithm originally inspired by biological evolution and Darwinian selection. A typical algorithm maintains a population of individuals (candidate solutions to a problem) that compete to survive in a simulated evolution. Solutions in the population are initially random and typically survive by maximizing some heuristic (Fogel, Owens et al. 1966). The algorithm utilizes stochastic operations inspired by biological evolution – such as mutation, recombination, and selection – to vary the population, recombine new individuals, and reward optimal solutions.

In a typical algorithm, each iteration (or generation) of the algorithm generates a successive population by selecting, crossing, and mutating individuals from the previous population. The selection process then picks individuals which perform the

best to be crossed and recombined with other individuals to create offspring for the next generation. Additionally, offspring undergo mutation which adds variation and diversity to the population. Mutation and crossover occur with some predefined probability. This allows some individuals to produce identical copies, mutated copies, crossed children, or crossed and mutated children.

Often, the best candidate solution in the population is tracked over each generation to measure progress. After the best solution has reached some desired level of performance, the solution is said to be converged, and the solution is returned.

### ***Symbolic Regression***

Symbolic regression is the problem of identifying the analytical mathematical description of a hidden system from experimental data (Augusto and Barbosa 2000; Duffy and Engle-Warnick 2002). Unlike polynomial regression or related machine learning methods that also fit data, symbolic regression is a system identification method, which attempts to reconstruct the representative structure of a system. Symbolic regression is closely related to general machine learning problems however, it remains an open-ended and discrete problem that cannot be solved directly.

Symbolic regression is an NP-hard problem, however, we can use an Evolutionary Algorithm to find solutions (Koza 1992; Schmidt and Lipson 2008; Schmidt and Lipson 2009). More specifically, the standard algorithm used in symbolic regression is genetic programming (Koza 1992), an evolutionary algorithm specialized for evolving computer programs and tree structures – for example, searching a space of mathematical expressions computationally and minimizing various error metrics. Both the parameters and the form of the equation are subject to search. In symbolic regression, many initially random symbolic equations compete to model experimental



data in the most parsimonious way. It forms new equations by recombining previous equations and probabilistically varying their sub-expressions. The algorithm retains equations that model the experimental data well while abandoning unpromising solutions. After an equation reaches a desired level of accuracy, the algorithm terminates, returning the most parsimonious equations that may correspond to the intrinsic mechanisms of the observed system.

In symbolic regression, the genotype or encoding represents symbolic expressions in computer memory. Often, the genotype is a binary tree of algebraic operations with numerical constants and symbolic variables at its leaves (McKay, Willis et al. 1995; Edwin and Jordan 2003). Other encodings include acyclic graphs (Schmidt and Lipson 2007) and tree-adjunct grammars (Nguyen, McKay et al. 2001). The fitness of a particular genotype (a candidate equation) is a numerical measure of how well it agrees with the data, such as the equation's correlation or squared-error with respect to the experimental data.

The operations can be unary operations such as *abs*, *exp*, and *log*, or binary operations such as *add*, *sub*, *multiply*, and *divide*. If some prior knowledge of the problem is known, the types of operations available can be chosen ahead of time (Augusto and Barbosa 2000; Soule and Heckendorn 2001; Duffy and Engle-Warnick 2002). The terminal values available consist of the function's input variables and the function's evolved constant values (Ferreira 2002).

Mutation in a symbolic expression can change an operator in the binary tree (e.g. change *add* to *sub*), change the arguments of an operation (e.g. change  $x+c$  to  $x+x$ ), delete an operation (e.g. change  $x+x$  to  $x$ ), or add an operation (e.g. change  $x+x$  to  $x+(x*x)$ ). If the operator is changed from a binary operation to a unary operation, for

example, one of the two child branches (chosen randomly) is discarded.

Crossover of a symbolic expression exchanges sub-trees in the binary trees of two parent expressions. For example, crossing  $f_1(x) = x^2 + c$  and  $f_2(x) = x^4 + \sin(x) + x$  could produce a child  $f_3(x) = x^2 + \sin(x)$ . In this example, the leaf node  $c$  was exchanged with the  $\sin(x)$  term.

## SECTION II – SEARCH METHODS

### CHAPTER 4. FITNESS PREDICTION

#### **Summary**

We present an algorithm that coevolves fitness predictors, optimized for the solution population, which reduce fitness evaluation cost and frequency while maintaining evolutionary progress. Fitness predictors differ from fitness models in that they may or may not represent the objective fitness, opening opportunities to adapt selection pressures and diversify solutions. The use of coevolution addresses three fundamental challenges faced in past fitness approximation research: (1) the model learning investment, (2) the level of approximation of the model, and (3) the loss of accuracy. We discuss applications of this approach and demonstrate its impact on the symbolic regression problem. We show that coevolved predictors scale favorably with problem complexity on a series of randomly generated test problems. Finally, we present additional empirical results that demonstrate that fitness prediction can also reduce solution bloat and find solutions more reliably.

#### **Introduction**

The chapter proposed the concept of fitness prediction – a technique to replace fitness evaluations in evolutionary algorithms with an exceedingly coarse approximation that adapts with the solution population. A closely related concept to fitness prediction is fitness modeling, where a predefined model or simulation is used to approximate fitness in cases where the exact fitness requires an expensive calculation or physical experiment (Jin, Olhofer et al. 2001; Ong, Nair et al. 2003). Fitness predictors however, cannot approximate the entire fitness landscape, but instead shift their focus throughout evolution.

Fitness approximations have been used in other situations as well, such as smoothing rugged fitness landscapes, mapping discrete fitness values to continuous values, and diversifying populations through ambiguity (Jin 2005). In this chapter we show that coevolving fitness predictors may also offer further benefits by destabilizing local optima and by resisting bloated solutions.

Recent research in fitness modeling and prediction has focused on approximation methods and strategies for use of approximated fitness values (Jin 2005). We review significant advances and challenges found in recent work and motivate a coevolutionary approach. We suggest that coevolution can resolve three fundamental difficulties faced in many fitness approximation applications:

- 1. Model training effort:** Often significant computational effort is required to train the desired fitness model.
- 2. Level of approximation:** It is often unclear what level of approximation is accurate enough to achieve desired results. High-quality approximations provide greater accuracy, but require more computation. Low-quality approximations are less accurate, but require less computation.
- 3. Loss of accuracy:** Similarly, even high-quality approximations are bound to have some loss of accuracy due to either the model structure itself or the data available to tune it. In the worst case, this effect can hide or even change the global optimum – in which case, exact fitness calculations are still needed to find the optimal solution.

The goal of this chapter is to address these issues through coevolution. In the general framework, there are three populations: (1) solutions to the original problem, evaluated using only fitness predictors, (2) fitness predictors of the problem, and (3) fitness trainers, whose exact fitness is used to train predictors. Solutions are evolved to

maximize their predicted fitness using a predictor from the predictor population. Fitness predictors are evolved to maximize prediction accuracy using trainers selected from the solution population. Trainers are evolved or selected to create discrepancies between predictors in order to address their weaknesses. Solution and predictor populations start with random solutions and random fitness predictors, respectively. The trainer population is initialized with random solutions and their exact fitness values.

In the following sections, we first review preliminary topics and current research in coevolution and fitness approximation. We then propose a coevolutionary algorithm based on a general framework and discuss its application in example domains. This algorithm is then adapted to the symbolic regression benchmark problem in genetic programming to measure its impact.

The experimental part of this chapter is structured as follows. First, we compare performance using three other fitness approximation methods to test what role coevolution plays in performance. We then duplicate experiments in recent symbolic regression literature and compare their results. We then test predictor performance as function of complexity on randomly generated target functions, in order to measure how the fitness prediction algorithms scale with respect to increasingly difficult problems. Finally, we discuss empirical trends demonstrating how coevolving fitness predictors can improve reliability and the quality of final solutions, even when the advantages of computational cost reduction are ignored.

## **Related Work**

### *Coevolution*

In a coevolutionary algorithm, the fitness metric for one individual becomes a function

of other individuals, possibly including itself. More precisely, one individual can affect the relative fitness ranking between two other individuals in the same or a separate population (Hillis 1992). As a result, the fitness pressures and incentives imposed on the solutions may change throughout evolution.

Coevolution is often applied to problems in which no explicit fitness objective is known in advance, or where the objective is abstract. For example, one may wish to find a solution that competes well against other solutions. In this example, competition between individuals imposed by coevolution can continuously expose weak individuals and refine successful individuals, until a dominant solution emerges perhaps.

Several studies have been devoted to the application of coevolution to enhance problem solving (Rosin 1997; Rosin and Belew 1997; Potter and De Jong 2000; Ficici and Pollack 2001; De Jong and Pollack 2004; Ficici 2004; Stanley and Miikkulainen 2004; Zykov, Bongard et al. 2005; Schmidt and Lipson 2006), with the main goal of controlling coevolutionary dynamics that often result in a lack of progress or progress in unanticipated directions (Cliff and Miller 1995; Pagie and Hogeweg 1997; Watson and Pollack 2001; Luke and Wiegand 2002; Bucci and Pollack 2005). Here we use a specific form of coevolution (Bongard and Lipson 2005; Bongard and Lipson 2005) which addresses many of these challenges.

The aim of coevolving fitness predictors is to allow both solutions and fitness predictors to enhance each other automatically until an optimal problem solution is found. The solution population benefits from the fitness predictor population through reduction in computational cost (and other benefits such as reduced bloat discussed later). The fitness predictor population benefits from the solution population by

refining its approximation in the most useful areas of the fitness domain.

### ***Fitness Modeling***

Fitness modeling has become an active area in evolutionary computation with many varying approaches and results (Jin 2005). Here we discuss the motivations, methods, and challenges of fitness modeling.

#### *Motivation*

There are several reasons for utilizing fitness approximation through modeling. The first, and most common, is to reduce the computational complexity of expensive fitness evaluations. However, approximation can be used advantageously in other problems as well. Fitness models have been applied to handle noisy fitness functions, smooth multi-modal landscapes, and define a continuous fitness in domains that lack an explicit fitness (e.g. evolving art and music) (Jin 2005). Here we discuss motivations for fitness modeling and example applications.

1. **Reducing complexity:** Many applications of evolutionary algorithms are in high-complexity or intractable domains where the fitness calculation can be prohibitively time consuming. For example, fitness modeling has been applied to structural design optimization (Jin, Olhofer et al. 2001; Jin, Olhofer et al. 2002; Mutoh, Nakamura et al. 2003; Ong, Nair et al. 2003; Jin and Sendhoff 2004; Regis and Shoemaker 2004; Regis and Shoemaker 2005) that often requires time-consuming finite element calculations. Often the resolution provided by the exact fitness objective is unnecessary for evolutionary progress.
2. **No explicit fitness:** Many domains do not have a computable fitness. For example, in human interactive evolution (Takagi 2001) (e.g. evolution of art

and music), a human user must select favorable individuals. Fitness models have been applied in these domains to reduce user fatigue and define a computable fitness landscape that can be searched while waiting for the user to give more feedback (Poli and Cagnoni 1997; Johanson and Poli 1998; Schmidt and Lipson 2006).

3. **Noisy fitness:** Some fitness functions are very noisy. To produce stable fitness rankings, algorithms typically average many evaluations, but this can greatly increase the computational cost (Arnold 2001). An alternative approach may be to develop a statistical model (Sano and Kita 2000).
4. **Smoothing landscapes:** Almost all evolutionary domains suffer from multi-modal landscapes that are often dense with local optima. Fitness approximation can greatly reduce the frequency and severity of local optima. Landscape smoothing has been observed with interpolation, kernels, and fitness clustering (Yang and Flockton 1995; Audet, Dennis et al. 2000; Regis and Shoemaker 2004; Regis and Shoemaker 2005).
5. **Promoting diversity:** When models smooth fitness landscapes, they often flatten local optima or produce different regions with similar fitness. While this is undesirable when using a single model throughout evolution, it can be advantageous for producing diversity as long as the fitness model continuously adapts, as is proposed in this chapter.

Despite their benefits, the use of fitness models can create new problems. Currently, it is not always clear when the benefits of fitness modeling outweigh the costs. In the following sections we overview basic fitness modeling approaches and their tradeoffs. We then discuss our approach to resolving these tradeoffs through coevolution.



### *Methods*

The technique of fitness modeling falls naturally in the field of machine learning. Depending on the structure of solution encodings, many different machine learning approaches such as neural nets, support vector machines, decision trees, Bayesian networks, k-nearest-neighbor, and polynomial regression can be trained to map individuals in order to approximate fitness values efficiently (Jin and Sendhoff 2004; Schmidt and Lipson 2006). Modern approaches utilize boosting, bagging, and ensemble learning to produce accurate models. A major drawback of these approaches is that it is often unclear which approach will work best for a given problem (Jin 2005).

Sub-sampling of training data is also a common way to reduce the cost of fitness evaluation (Pagie and Hogeweg 1997; Albert and Goldberg 2002). In many problems, fitness is calculated by evaluating individuals on training cases and combining the total error. With a sub-sample, only a fraction of the training data is evaluated.

Evolutionary-specific fitness modeling methods include fitness inheritance, fitness imitation, and partial evaluation. In fitness inheritance (Smith, Dike et al. 1995; Sastry, Goldberg et al. 2001; Chen, Goldberg et al. 2002), fitness values are transferred from parents to children during crossover (similar to parent passing on a legacy or education). A form of fitness inheritance for estimation of distribution algorithms (Larrañaga and Lozano 2002) (EDAs) builds a model of the fitness function based on the structure of the probabilistic model used in the algorithm (Pelikan and Sastry 2004). In fitness imitation (Jin and Sendhoff 2004), individuals are clustered into groups based on a distance metric. The fitness of the central individual of each cluster is then evaluated in full and assigned to all individuals in that cluster. In partial evaluation (Ochoa and Soto Ortiz 1997), the fitness of some individuals are calculated

exactly, while others are modeled or inherited.

Once a fitness model has been chosen, there are many ways to incorporate it into the evolutionary process. It can be used simply to initialize the population, guide crossover and mutation, or replace (some) fitness evaluations (Jin 2005). For example, a fitness predictor such as a neural network is used to select offspring from all potential crossovers of two parents (Mutoh, Nakamura et al. 2003). In this chapter however, we focus only on replacing actual fitness evaluations with the fitness predictor.

### *Challenges*

The use of an approximate fitness model comes at a cost and with potentially unacceptable consequences.

- 1. Training the model:** Fitness models like neural nets, SVMs, and Gaussian processes require significant overhead to train. When advanced methods like bagging, boosting, and ensemble methods are used, this investment becomes significantly larger. In addition, a significant amount of exact fitness values must be calculated for training and validation data to effectively learn any type of model ahead of time.

By using coevolution, we can train these models in parallel with the problems' solutions. As shown in (Yang and Flockton 1995), early stages of evolution only require coarse fitness models. As the solution population progresses, so do the fitness models. In this fashion, coevolution retains an automatic 'coarseness adjustment' without the need to train several different approximations in advance.

- 2. Level of approximation:** How powerful must the fitness model be to facilitate

progress throughout evolution? If a single fitness model is used, it may need to be quite complex in order to model all possible solutions in the fitness landscape.

When fitness models are coevolved, the models can be optimized for only the individuals in the current population. The models do not need to encapsulate the entire landscape, but only a subset, so the chosen method can be significantly less complex.

- 3. Loss of accuracy:** In most applications, the computational advantage of using a fitness approximation comes at a cost in fitness accuracy. In the worst case, the global optima may be removed entirely from the fitness landscape.

Similar to adapting the level of approximation, the optimization of the models to the current population can keep the subjective fitness of current candidate solutions pointed toward the global optima in an active learning fashion (Bongard and Lipson 2005). Solutions will evolve to exploit their fitness model. In coevolution, the fitness model can adapt through the selection of trainers to redirect solutions so that they are consistent with the true optima.

## **Fitness Prediction Algorithm**

### ***General Framework***

In this section we present a simple framework before describing our implementation. A conventional evolutionary algorithm can be viewed as an optimization to find the most fit solution. In this sense, the optimal solution,  $s^*$ , is defined as:

$$s^* = \arg \max_{s \in S} \text{fitness}(s)$$

where  $S$  is the set of all possible candidate solutions to the problem and  $\text{fitness}(s)$  is the

exact computed fitness of solution  $s$ .

In the coevolutionary algorithm, we replace all fitness evaluations with a fitness predictor,  $p$ . In this instance, the solution objective is a function of the predictor instead of the exact fitness:

$$s^* = \arg \max_{s \in S} p(s)$$

where  $p$  is the fitness predictor used.

We coevolve the fitness predictors in a second population to make  $p$  as accurate as possible for the current solution population. A third population of fitness trainers is used to evaluate how closely fitness predictors are approximating the exact fitness. Fitness trainers are chosen from the solution population periodically that have the highest prediction variance (e.g. lowest confidence).

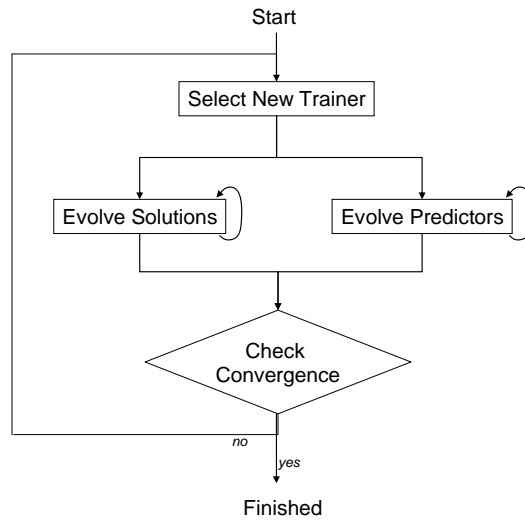
The objectives for each population are summarized below, where asterisks specify an optimal result that is being searched for in each population.

$$s^* = \arg \max_{s \in S} p_{best}(s) \quad (\text{Solutions})$$

$$t^* = \arg \max_{s \in S_c} \frac{1}{N} \sum_{p \in P_{cur}} \left( p(s) - \overline{p(s)} \right)^2 \quad (\text{Trainers})$$

$$p^* = \arg \min_{p \in P} \frac{1}{N} \sum_{t \in T_{cur}} |fitness(t) - p(t)| \quad (\text{Predictors})$$

where  $S$  is the set of all problem solutions,  $S_c$  is the current solution population,  $P$  is the set of all possible fitness predictors,  $P_{cur}$  is the current predictor population,  $T_{cur}$  is the current trainers population,  $p_{best}$  is the highest ranked predictor in  $P_{cur}$  and  $\overline{p(s)}$  is



**Figure 4.1. High-level overview of the coevolution of solutions and fitness predictors algorithm.**

the average predicted fitness of solution  $s$  among the current predictors. It is important to note that all three populations are evolved in parallel and their objectives will be dynamic and changing over each generation.

To summarize the framework, the solution population evolves to maximize the fitness of the current best fitness predictor. Trainers are solutions chosen from the solution population that produce the most variance in predictions among the predictor population. The fitness predictor population evolves to minimize the difference between exact and predicted fitness values of the current population.

### ***Algorithm***

#### *Summary*

The algorithm presented in this chapter has three populations: Problem solutions, fitness predictors, and fitness trainers. This section outlines the basics needed to implement this coevolutionary approach based on this general framework. A high-

level algorithm overview is given in Figure 4.1.

At the start, solutions, fitness predictors, and trainers are randomly initialized. The algorithm then chooses an individual from the solution population to measure its exact fitness for use in training the fitness predictors (elaborated upon in next section). The algorithm then evolves the solution population using the highest ranked fitness predictor, and evolves the predictors using the fitness trainers. Finally, the highest-ranked individual is tested for convergence (described below), and the algorithm completes if successful. Pseudocode for evolving each population is provided in Figure 4.2.

#### *Evaluating Exact Fitness Values*

The objective of this step is to select an individual from the solution population that will help the fitness predictors optimize to the current solutions. Therefore, we want to choose an individual whose fitness can be predicted with the least confidence. To do this efficiently, we select the individual that has the highest variance in predicted fitness among predictors in the predictor population. Variance has a strong correlation with reducing uncertainty (Jin and Branke 2005) and with improving evolved individuals (Bongard and Lipson 2005).

In many model types, it is often beneficial to “forget” past solution fitness information in order to allow simple predictor encodings to specialize in only the current and other recently observed solutions. In our implementation, we store only the most recent trainers, discarding the oldest as new trainers are evaluated.

Removal of old trainers can also speed up predictor evaluation, but could lead to cycling. For example, removing a trainer may remove pressure to explain an important part of the fitness domain. In which case, solutions and predictors that modeled this

region well could drift away temporarily while learning other regions. To prevent this effect, we could opt to keep all trainers for an additional computational cost – but we did not find cycling to be prohibitive in our experiments.

### *Evolving the Populations*

Candidate solutions and fitness predictors are coevolved in parallel using two threads. Pseudocode is shown in Figure 4.2. Fitness trainers are selected periodically in the predictor thread.

The solution thread begins by randomizing the population of candidate problem solutions. The main loop then evolves the solution population. Variation is introduced using single point crossover with probability  $p_c$  and mutation with probability  $p_m$ . The highest ranked fitness predictor is then used to estimate the fitness of each child and selected to form the next generation. Finally, the top ranked solution is tested for convergence (described in the next section) and exits.

The predictor thread begins by randomizing the fitness predictor and fitness trainer populations. The main loop then evolves the predictors and periodically adds new trainers to the trainer population. Variation is introduced using single point crossover with probability  $p_c$  and mutation with probability  $p_m$ . The fitness of each predictor is calculated by the mean absolute error between the fitness prediction and the exact fitness for each fitness trainer.

---

---

**Solutions Thread:**

```
Randomize solution population
Repeat
  Cross solutions with probability  $p_c$ 
  Mutate solutions with probability  $p_m$ 
  Let pred = the top ranked fitness predictor
  Predict fitness values for solutions using pred
  Perform selection
  Sort population
  If top-ranked solution error < epsilon
    Return solution and Exit
  End if
End repeat
```

---

---

**Predictor Thread:**

```
Randomize predictor population
Randomize trainer population
Repeat forever
  If computational effort > 5% of total
    Wait
  End if
  Cross predictors with probability  $p_c$ 
  Mutate predictors with probability  $p_m$ 
  Evaluate fitness values of predictors
  Perform selection
  If time to add new fitness trainer
    Let  $v_i$  = the variance in fitness
      predictions of all predictors for
      solution i
    Add solution i with the highest  $v_i$  to
      the trainer population
    Calculate the exact fitness of the new trainer
  End if
End repeat
```

---

---

**Figure 4.2. Pseudocode for the two threads in the algorithm that coevolve solutions and predictors. Trainers are chosen periodically in the predictor thread.**

Lightweight fitness predictors tend to evolve much faster than the solutions and therefore do not require as much computational effort. To reduce computational effort,



we artificially slow evolution of the predictor population by introducing a delay. If the computational effort (measured in point evaluations<sup>1</sup>) used to evolve the predictors exceeds some percentage of the total effort of all populations (5% in our experiments), the predictor thread is delayed. The specific choice of effort allocation is likely problem-dependent; however, we have observed that the 5% ratio performs well over a relatively wide range of values (as shown in the results section below).

New fitness trainers are chosen from the solution population periodically. Fitness trainers are solutions that the fitness predictors optimize to predict. In our implementation, we choose a new trainer to add to the trainer population every 100 fitness predictor population generations. This augmentation of the trainer population provides time for the fitness predictors to adjust their approximation and is related to the speed at which predictors converge. Alternatively, new trainers could be selected continuously, or whenever the progress of the predictor population slows.

### *Convergence Test*

The convergence test determines when the algorithm should terminate by testing the solution in the current population that has the highest predicted fitness. For symbolic regression, we define convergence as having near zero ( $<\epsilon$ ) error on all training data examples. If the best solution has not converged at this step, a new trainer is added (Figure 4.1) and evolution continues; otherwise, the best solution is returned and the program terminates. As in any machine learning algorithm, the final solution performance must be cross-validated against an unseen test set.

---

<sup>1</sup> Here and elsewhere in this chapter, we measure performance as function of number of point evaluations, instead of number of generations or number of fitness evaluations. We use this metric in order to perform fair comparisons between methods that use different computational efforts per evaluation.

## Experiments in Symbolic Regression

We evaluated our proposed approach using symbolic regression as an example application of fitness predictor coevolution. Symbolic regression serves as a good benchmark since it is a well-studied domain with diverse applications.

We first experiment on simple functions, then duplicate experiments from recently published research, and finally experiment on thousands of randomly generated symbolic target functions of increasing complexity.

We generated random target functions by building a random tree of operations. The

---

**Initialize:**

```
Func = binary tree of random depth [1,12]
Func.Randomize_Operators()
Func.Remove_Random_Child_on_Unary_Operators()
```

**Branch Prune:**

```
Test = Func
For each Node1, Node2:
  Test.Remove(Node1, Node2)
  If Max_Output_Difference(Func, Test) < EPSILON:
    Func = Test
  Else:
    Test = Func
End for
```

**Node Prune:**

```
Test1 = Test3 = Test3 = Test4 = Func
For each Node1, Node2:
  For each Child1 in Node1 and Child2 in Node2:
    Test1.Node1 = Node1.Child1
    Test1.Node2 = Node2.Child2
    If Max_Output_Difference(Func, Test1) < EPSILON:
      Func = Test1
  End for
End for
```

---

**Figure 4.3. Pseudocode for pruning inactive expressions in randomly generated test problems to improve the complexity estimate for problem difficulty.**

tree is binary, with the exception of unary operators which only have a single child. We then prune combinations of nodes in the function's tree that result in less than  $\varepsilon = 1\%$  change in function output across a target range (between -2 and 2 inclusive for our experiments), using the code below. We define the complexity of the resulting function as the number of nodes in the pruned tree. Example randomly generated functions and their respective complexities are shown in Table 4.3.

### ***Symbolic Regression Overview***

#### *Symbolic Regression Encoding*

For experiments in this chapter, we represent functional expressions as a binary tree of primitive operations (Koza 1992; Augusto and Barbosa 2000; Eggermont and Hemert 2000). See the description in the section “Symbolic Regression” on page 4 for more detail.

The fitness objective of a symbolic regression solution is to minimize error on the training set (Eggermont and Hemert 2000; Dolin, III et al. 2002; Hoai, McKay et al. 2002; Keijzer 2003). There are many ways to measure the error such as squared error, absolute error, etc. For experiments in this chapter, we use the mean absolute error for fitness measurement:

$$fitness(s) = \frac{1}{N} \sum_{i=1}^N |s(x_i) - y_i|$$

where  $s(x_i)$  is output of a candidate solution  $s$  evaluated on input  $x_i$ , the value  $y_i$  is the corresponding output, and  $N$  is the number of training examples in training data set.

#### *Coevolution in Symbolic Regression*

Coevolving training examples is a well-studied approach in symbolic regression

(Pagie and Hogeweg 1997; Dolin, III et al. 2002). Past research has competitively coevolved training examples to exploit errors, an approach similar to boosting methods in machine learning. Coevolving examples to diversify solutions and moderate purely competitive pressures have also been studied.

Very little work, however, has been done in fitness prediction or modeling in symbolic regression. In our experimentation, we coevolve a subset of the total training data examples that approximates fitness measurement over the complete training data. The set's objective is to guide evolution as closely as possible to using the entire training set, but at a reduced computational cost.

### ***Sub-sample Fitness Predictors***

#### *Fitness Predictor Encoding*

Training data in symbolic regression typically consists of hundreds to thousands of data points (e.g. experimental measurements) providing output values for a sample of inputs. In our symbolic regression experiments, the fitness predictor is a small subset of these points. Instead of measuring the exact objective fitness of candidate solutions, a subjective fitness is obtained by measuring the error on the select handful of data points of a given fitness predictor.

The fitness predictor is encoded as a small array of indexes to the full training data set (size discussed in the next section). Each index in the predictor's array is free to reference any points in the training data examples and can repeatedly sample point if it likes (thus over emphasizing an area). The predicted fitness is calculated as:

$$predicted\_fitness(s) = \frac{1}{n} \sum_{i=1}^n |s(x_i) - y_i|$$

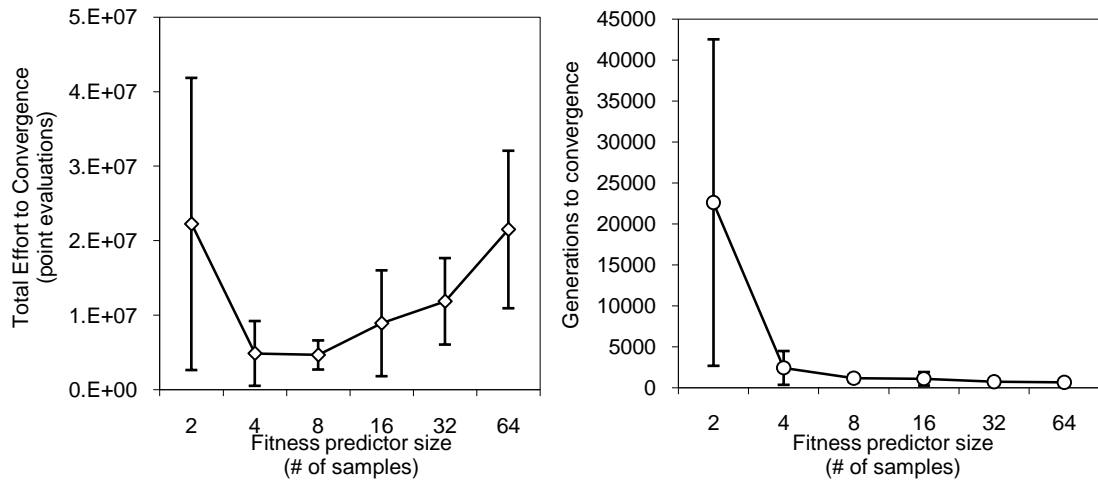
Here,  $n$  is the number of samples in the predictor, and symbols  $s$ ,  $x_i$ , and  $y_i$  are the same as above.

Mutation in the fitness predictor can randomize an index in its array to index a different training point. An example point mutation would be (1, 41, 53, 92) changing to (1, 78, 53, 92), where the sample 41 switched to 78.

Crossover exchanges the samples of two parent fitness predictors. For our purpose, we use a single point crossover. A random crossover point  $c$  is chosen, the first  $c$  indexes are copied from the first parent and the remaining indexes are set from the second parent.

#### *Size and Complexity of the Fitness Predictor*

There is an inherent tradeoff between predictor size (subset size) and overall performance. Using a small number of samples in the fitness predictor allows for more generations while maintaining the same computational effort, at the cost of less accurate prediction. We empirically examined the sensitivity of the number of samples in the training subset fitness predictor using an arbitrary function  $f=e^{|x|}\sin(x)$ . This function is a simple non-linear function that has two local minima approximations that make finding the exact solution difficult. In following sections we also use this function as a benchmark for some empirical experimentation because, although it evolves rapidly, it is clearly non-trivial.



**Figure 4.4. The expected point evaluations before convergence versus the number of samples in the fitness predictor. Error bars show the standard deviation.**

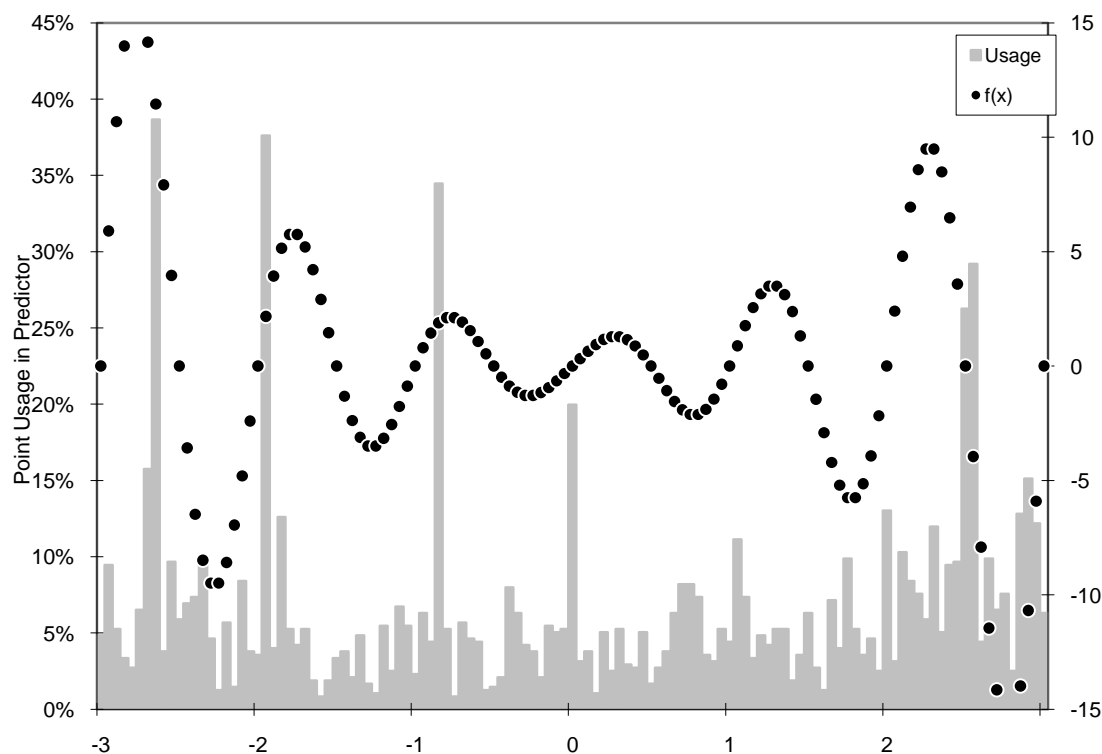
When the fitness predictor only has two samples, fitness evaluations are extremely light-weight but the evolutionary process requires many more generations, as evident in Figure 4.4. The larger subsets are sufficiently large for accurate modeling but do not greatly reduce the number of generations needed. Figure 4.4 also suggests that there is some minimum number of samples needed for a given target function or the available training data. We hypothesize that the optimal number of samples is higher for complex functions with more detailed features, but we have yet to see this number increase dramatically even with high complexity functions (over 30 nodes in the expression tree) as tested later in this chapter.

In our symbolic regression experiments, we use an 8-sample subset for all experiments. Although it may not be the optimal choice for all target functions, these results suggest that it will not have a dramatic impact on final performance. Varying the number of samples from eight did not appear to have a strong impact on the performance on several other target functions tested, even in the cases of high complexity multi-variable functions involved in on-going research.

### *Fitness Predictor Behavior*

Here we preview how fitness predictors may behave in symbolic regression. The fitness predictors used here are small subsets of the training set and are optimized by trainers chosen from the solution population. Thus, the types of subsets evolved are determined by how the solutions evolve and are likely to vary over different problems and even different runs. However, a few empirical trends can be seen in this type of fitness predictor.

Figure 4.5 shows a histogram of the training points used by the best fitness predictor up to convergence on the function  $f=e^{|x|}\sin(x)$ . For this run, there are seven highly used training points which are used in 20% to 40% of generations up to convergence. Notice that the most used points tend to lie to the sides of local minima and maxima in



**Figure 4.5.** Histogram of training samples selected by the best fitness predictor during evolution to convergence of  $f(x)=e^{|x|}\sin(x)$ . Some samples are selected significantly more often than others.

the training data. This may indicate an effective way to capture features of the dataset without overestimating the averaged error. In particular to this function, these points may be necessary to fine-tune candidate solutions to match the function's periodic structure.

### ***Experimental Settings***

For each independent run, all symbolic regression parameters were held constant, and only the type of predictor was varied. We used a solution population size of 128, a fitness predictor population size of 8, and a trainer population size of 10. For evolution we use deterministic crowding selection (Mahfoud 1995), 0.1 mutation probability, and 0.5 crossover probability.

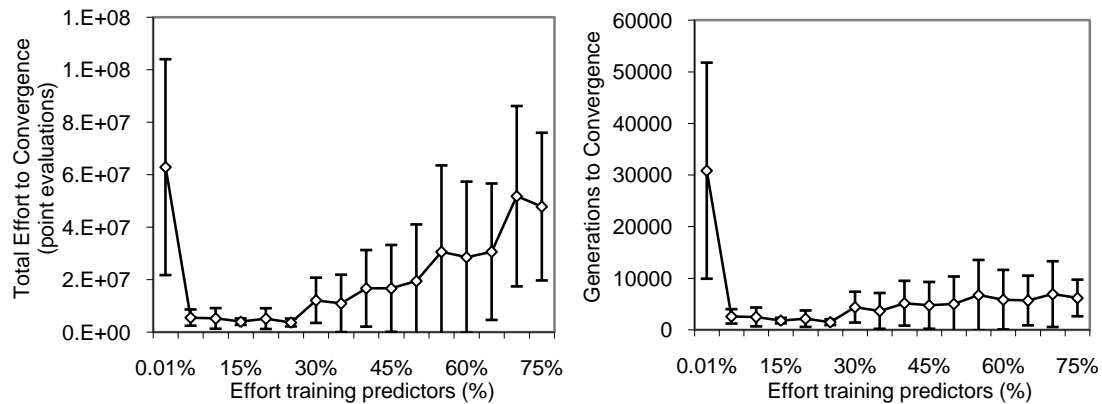
The operator set was (*add, subtract, multiply, divide, exponent, logarithm, sine, cosine*) and the terminal set consists of the input variable and one evolved constant. In practice, *a priori* knowledge could be applied to choose a more useful operator and terminal sets. For example, the experimenter may not be interested in expressions that use many evolved constants, or solutions that involve trigonometric functions. However, in our experiments, we use the same parameters throughout testing and the terminal and operator sets are over-representative for all targets (e.g. more operators are available than needed to regress the function).

### ***Computational Effort Distribution Among Populations***

For experimental purposes, we control how much effort is spent training the fitness predictors in relation to the solutions so that we can compare algorithms based on their total overall computational effort. Note that in practice, the ratio is not vital to the algorithm's performance since each population can be evolved in parallel.

Figure 4.6 shows the impact that the effort ratio has on convergence time with the test





**Figure 4.6. The expected number of point evaluations before convergence versus the effort (percent of point evaluations) while training the fitness predictors averaged over 50 trials. Error bars show the standard error.**

function  $f=e^{|x|}\sin(x)$ . Ratios in the range 5% to 30% of effort spent training the fitness predictor population all yield approximately optimal convergence time. If fitness predictors are given extremely low computational effort, overall performance suffers greatly since the fitness approximation never adapts.

Spending excessive effort training fitness predictors tends to add no extra benefit. The computational effort increases, but solution generations remain approximately the same.

In summary, the fitness predictors need some minimal amount of effort so that they are able to adapt with the solutions. Thus, the relative rates of evolution need be considered before choosing a minimal effort ratio so that they have similar time-scales. Since fitness predictors are expected to be simple and light-weight, they should require only a fraction of the effort that the solutions require.

## Experimental Results

### *Examining Behavior on Test Problems*

Here we compare four fitness algorithms in symbolic regression listed in Table 4.1.

**Table 4.1. Summary of the Compared Algorithms**

<b>Fitness Calculation</b>	<b>Sample Size</b>	<b>Sample Selection</b>
Coevolved Predictor Sample	8	Evolved subset
Static Random Sample	8	Random subset chosen at runtime
Dynamic Random Sample	8	Changing random subset
Exact Fitness	200	Use all training data

These algorithms are used as null hypotheses to elicit the effect of coevolution.

The Static Random Sample algorithm uses a single fitness approximation throughout evolution. Eight random samples are chosen from the training data at run time, and solutions are evolved using only this sample. This algorithm tests the hypothesis that the performance improvement is made simply from reducing point evaluations.

The Dynamic Random Sample algorithm is similar to the Static algorithm, but now the sample is re-randomized at every generation of the solutions. This algorithm tests the hypothesis that performance improves not only because of reducing point evaluations but also because of allowing the sample to change.

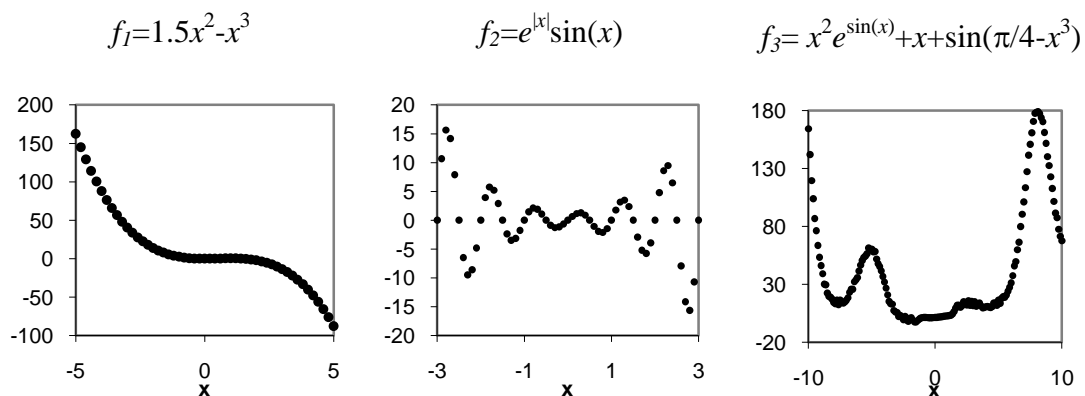
The Exact Fitness Algorithm is given for the purpose of baseline comparison. The solutions are evolved using the exact objective fitness, as is usually practiced in symbolic regression research (Eggermont and Hemert 2000; Dolin, III et al. 2002; Hoai, McKay et al. 2002; Keijzer 2003).

In this section, we test on three different target functions that elicit different behaviors from the four algorithms. The training data, shown in Figure 4.7, are 200 evenly spaced samples of the target function. The test set contains 200 additional random samples. Each experiment is repeated 50 times independently, and the fitness for each run is recorded over evolutionary time.

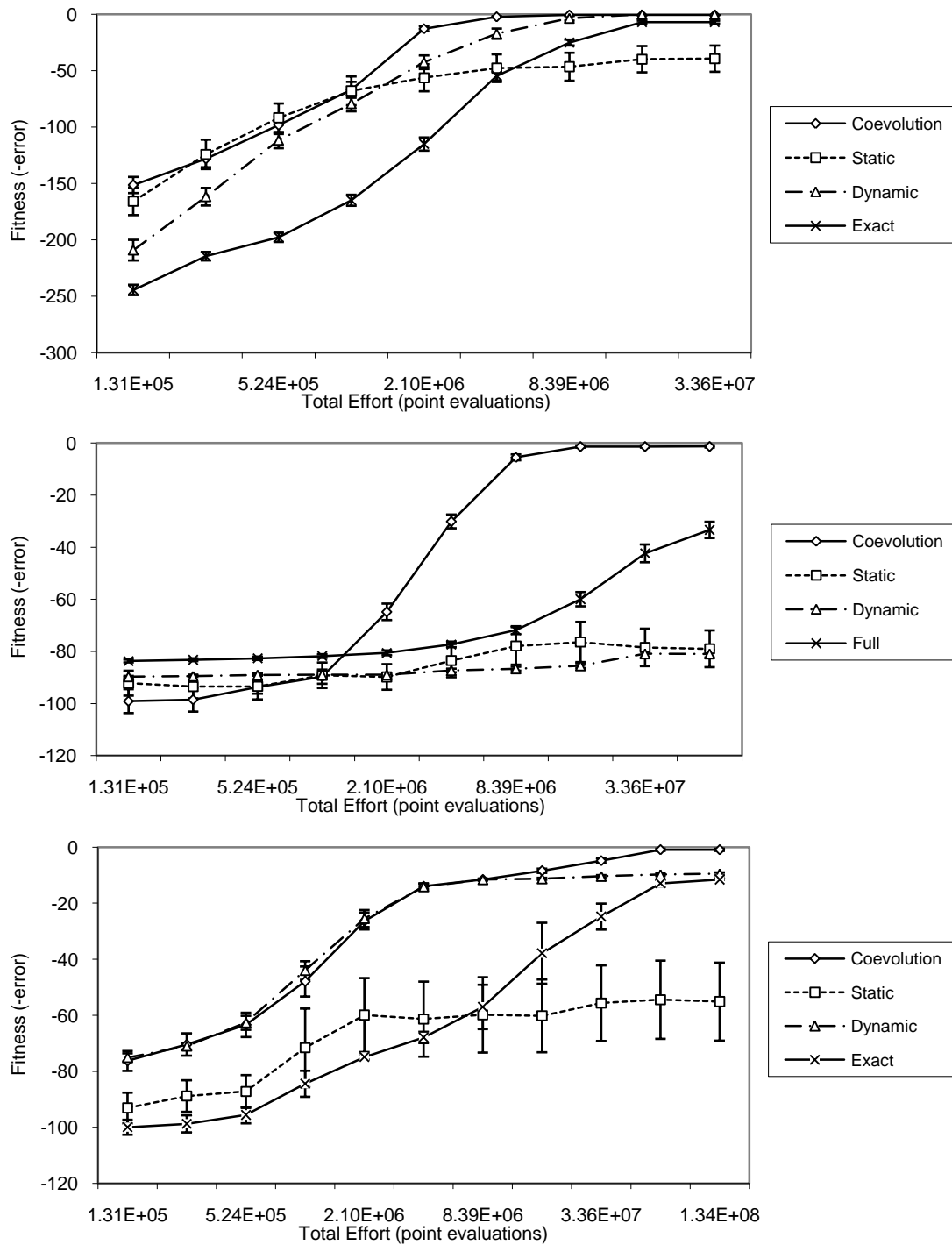
The performances on these three functions for each algorithm are shown versus the number of point evaluations in Figure 4.8.

The polynomial function  $f_1(x)$  is very simple and coevolution, static random, and exact fitness all rapidly converge. The coevolution and static random methods make similar improvements over exact fitness, suggesting that the improvement is chiefly due to the reduction in function evaluations.

Behavior on  $f_2(x)$  is different however. The static and dynamic random sample algorithms perform very poorly on average, and the exact fitness algorithm outperforms them. However, coevolution still makes a substantial improvement over exact fitness.



**Figure 4.7.** The training data of the three target functions experimented on. The horizontal axis shows the input values  $x$ . The vertical axis shows the output training value  $f(x)$ .



**Figure 4.8.** The test set fitness during evolution for target functions  $f_1(x)$ ,  $f_2(x)$ , and  $f_3(x)$  respectively. Results are averaged over 50 trials. Error bars show the standard error.

In contrast, function  $f_3(x)$  gives an example in which the dynamic random sample performs very well. It is able to find the large features of the function as quickly as coevolution; however, it fails on the final sine feature.

We can make several conclusions from these results. First, the static random sample shows performance can be improved on a simple function like  $f_1(x)$  simply by using a small subset for fitness calculation. On more complicated functions however, a small constant subset alone cannot adequately represent features of more complicated functions like  $f_2(x)$  or  $f_3(x)$ .

Conversely, the dynamic random sample algorithm can greatly improve performance on some more complicated functions such as  $f_3(x)$ . Using a sample that changes randomly can accelerate finding large features of the data but may fail on simple features as in  $f_1(x)$ ,  $f_2(x)$ , or the sine term in  $f_3(x)$ .

For these basic test cases, coevolution performs the best in each case. We can reject the hypotheses that the performance improvement is due only to using a sub-sample or a randomly changing sub-sample. Thus, the effect of coevolution must play an important role. Later in this chapter we compare the convergence rates of these algorithms over randomly generated functions to observe more general trends.

### ***Comparison to Previously Published Methods***

In this section, we compare the coevolution algorithm with four recently published symbolic regression techniques: Stepwise Adaptive Weights (SAW) (Eggermont and Hemert 2000), Grammar Guided Genetic Programming (GGGP) (Hoai, McKay et al. 2002), Tree-Adjunct Grammar Guided Genetic Programming (TAG3P) (Hoai, McKay et al. 2002), Coevolution with Tractable Shared Fitness (Dolin, III et al. 2002), Distinction Fitness (Dolin, III et al. 2002), and random sampling (Dolin, III et al.

2002). We did not re-implement these algorithms. Instead, we ran our algorithm on the same test problems reported in the original papers, using the same convergence criteria used in the original paper.

We compare computational performance based on point evaluations, defined by the total number of times the output of any symbolic expression is evaluated. The coevolution algorithm is stopped based on the number of point evaluations that the compared algorithm made during each experiment. In the compared algorithms, we assume that each individual's fitness is measured every generation. Likewise, we force the coevolution algorithm to calculate fitness for every generation, even though different selection algorithms do not require it.

Many of these experiments are on simple functions but are stopped at a very low number of point evaluations. Thus, finding the target function quickly is the highest priority. The cosine identity and the Gaussian function experiments are noticeably more challenging to regress based on parameters specific to these experiments.

Qualitative improvements in Table 4.2 are shown in bold text. The coevolution algorithm has slightly higher convergence than the PSAW and GGGP algorithms on polynomial problems. The TAG3P algorithm performs the best on simple polynomials; however, there is a qualitative difference when applied to a harder problem: regressing the double angle cosine identity. Coevolution makes a 40% improvement in convergence for the trigonometric identity experiment. The comparison with coevolved tractable, shared, and random sampling algorithms show coevolution can make substantial improvements in regressing a Gaussian function, traditionally a very challenging problem for symbolic regression in which over 90% of the data points lie on the tail (Dolin, III et al. 2002).

**Table 4.2. Performance comparison to published methods**

Algorithm	Target Function <sup>§</sup>	Metric <sup>§</sup>	Published Results	Coevolved Predictors
PSAW	$f(x) = x^5 - 2x^3 + x$	Convergence <sup>†</sup>	85.9%	<b>93.9%</b>
	$f(x) = x^6 - 2x^4 + x^2$	Convergence <sup>†</sup>	81.8%	<b>86.9%</b>
GGGP	$P2, P3, P4, P5^*$	Convergence <sup>††</sup>	92%, 64%, 48%, 28%	<b>100%, 86%, 62%, 52%</b>
	$f(x) = \cos(2x)^{**}$	Convergence <sup>††</sup>	20%	<b>76%</b>
TAG3P	$P2, P3, P4, P5^*$	Convergence <sup>††</sup>	100%, <b>100%</b> , <b>96%, 84%</b>	100%, 86%, 62%, 52%
	$f(x) = \cos(2x)^{**}$	Convergence <sup>††</sup>	36%	<b>76%</b>
Coevolved Tractable	Gaussian	Evaluations <sup>†††</sup>	3.384e7	<b>2.107e7</b>
Coevolved Distinction	Gaussian	Evaluations <sup>†††</sup>	5.070e7	<b>2.107e7</b>
Random Sampling	Gaussian	Evaluations <sup>†††</sup>	6.006e8	<b>2.107e7</b>

\* P3, P4, P5 etc. refer to polynomials ( $x^3+x^2+x$ ,  $x^4+x^3+x^2+x$ ,  $x^5+x^4+x^3+x^2+x$ , ... )

\*\* The operator set does not include the cos() function, a trigonometric identity must be found

† The percent of successful convergences from 100 test runs

†† The percent of successful convergences from 50 test runs

§ This target function and metric was used in the original paper

††† The maximum number of evaluations before convergence for 100 test runs

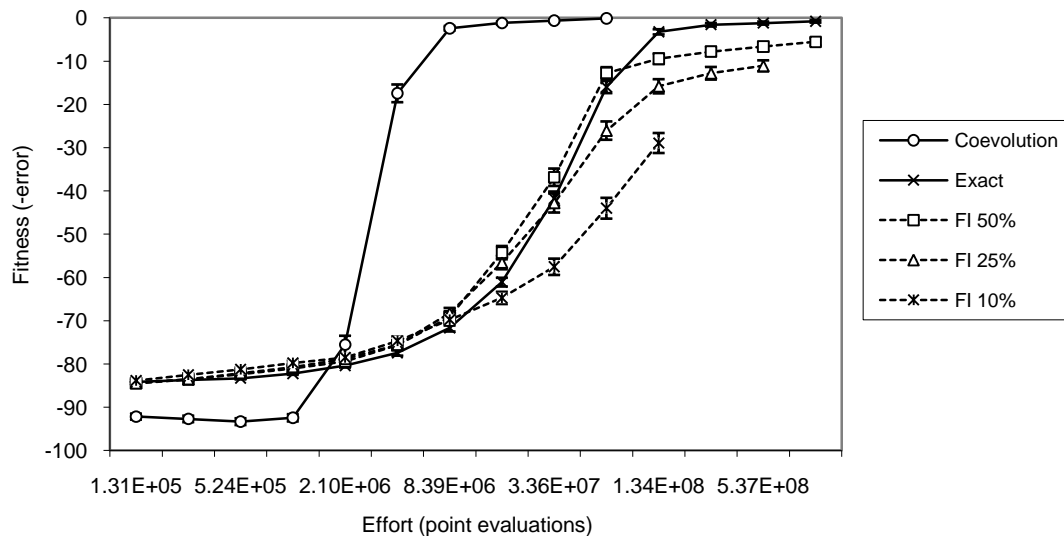
Next, we make an empirical comparison with fitness inheritance (Smith, Dike et al. 1995; Sastry, Goldberg et al. 2001; Chen, Goldberg et al. 2002). As mentioned in above, fitness inheritance is a fitness modeling approach that evaluates exact fitness values for a fraction of the population and allows the inheritance of fitness values during crossover for remaining individuals. We implemented fitness inheritance in symbolic regression by tagging 10%, 25%, and 50% of individuals each generation to

use exact fitness calculations and the rest to use their inherited fitness (or last exact fitness).

Figure 4.9 compares performance by the computational effort. In this experiment, runs were stopped after 20,000 generations. Exact fitness and fitness inheritance use more point evaluations and therefore show more data points on the plot.

Fitness inheritance appears to behave very similarly to the exact fitness algorithm in symbolic regression. Using 50% exact evaluations in fitness inheritance does accelerate over exact fitness on several runs; however, further attempts to reduce evaluations worsen the average performance.

This result is consistent with other work involving fitness inheritance. In related work (Jin, Olhofer et al. 2002), the authors conclude that 50% of fitness evaluations need to be based on exact fitness to ensure reliable convergence. In contrast, fitness prediction distributes a small fraction of point evaluations to estimate the fitness of all individuals



**Figure 4.9.** Test set fitness versus evaluations averaged over 100 test runs for  $f_2(x)$ . Error bars show standard error.



**Table 4.3. Example functions and complexities**

<b>Random Function</b>	<b>Complexity</b>
$f(x) = x$	1
$f(x) = x^2 - x$	5
$f(x) = \sin(\cos(x)) \cdot (\exp(x) - \cos(x))$	11
$f(x) = \exp(( x  + \exp(x)) / ((\exp(x) + \sin(x)) -  (x/x) ))$	23
$f(x) = \log(\cos(x + (\exp(\sin(x) \cdot  x ) \cdot (\sin(x \cdot \log(x)) + \exp(\cos(x)))))))$	37

in every generation, the equivalent of roughly 5% full evaluations per generation in this experiment. This demonstrates that a compromise between exact fitness evaluations and approximated fitness values can yield performance increases with similar convergence rates.

#### ***Testing Scalability on Randomly Generated Test Problems***

The experiment presented in this section explores the behavior of the coevolution algorithm when solving for randomly generated functions of varying complexity.

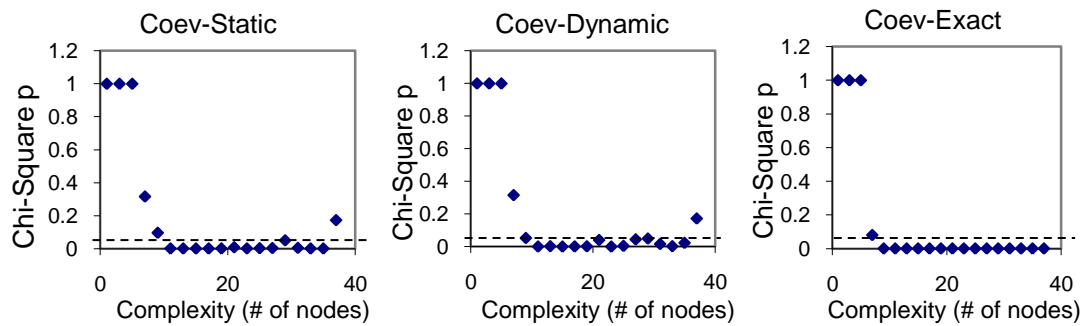
We generate random target functions by building a random binary tree of operations. We then perform a rough simplification by systematically pruning combinations of nodes in the function's binary tree and then testing for a significant change in the functions' outputs (see Appendix A). Next, the function is evenly sampled 100 times over the range [-2, 2] to generate the training data and then randomly sampled to produce the test set.

**Table 4.4. Chi-Square Significance of Convergence Rates Compared to the Coevolution Algorithm**

Complexity	Chi-Square p-value		
	Static	Dynamic	Exact
1	1	1	1
3	1	1	1
5	1	1	1
7	0.315692	0.315692	0.080181
9	0.095581	0.052926	<b>1.54E-05</b>
11	<b>1.08E-05</b>	<b>0.000536</b>	<b>1.96E-10</b>
13	<b>9.56E-06</b>	<b>0.002441</b>	<b>7.75E-17</b>
15	<b>4.1E-07</b>	<b>0.000281</b>	<b>9.57E-18</b>
17	<b>3.92E-05</b>	<b>0.001073</b>	<b>5.17E-20</b>
19	<b>0.000431</b>	<b>0.001726</b>	<b>4.75E-32</b>
21	<b>0.007439</b>	<b>0.040599</b>	<b>2.57E-34</b>
23	<b>0.000303</b>	<b>0.000303</b>	<b>4.84E-25</b>
25	<b>0.001503</b>	<b>0.004607</b>	<b>4.32E-16</b>
27	<b>0.002755</b>	<b>0.044423</b>	<b>1.91E-13</b>
29	<b>0.049535</b>	<b>0.049535</b>	<b>1.19E-09</b>
31	<b>0.003649</b>	<b>0.0161</b>	<b>2.14E-23</b>
33	<b>1.71E-19</b>	<b>0.002359</b>	<b>1.71E-19</b>
35	<b>1.23E-08</b>	<b>0.022948</b>	<b>1.23E-08</b>
37	0.172386	0.172386	<b>1.94E-05</b>

We define the “complexity” in this experiment to be the number of nodes in the generating target function. Examples of randomly generated functions and their respective complexities are shown in Table 4.3.

We generate 5000 random target functions for this experiment in order to produce training and test datasets of various complexities. Functions are uniformly spaced on odd-numbered complexities from 1-40.



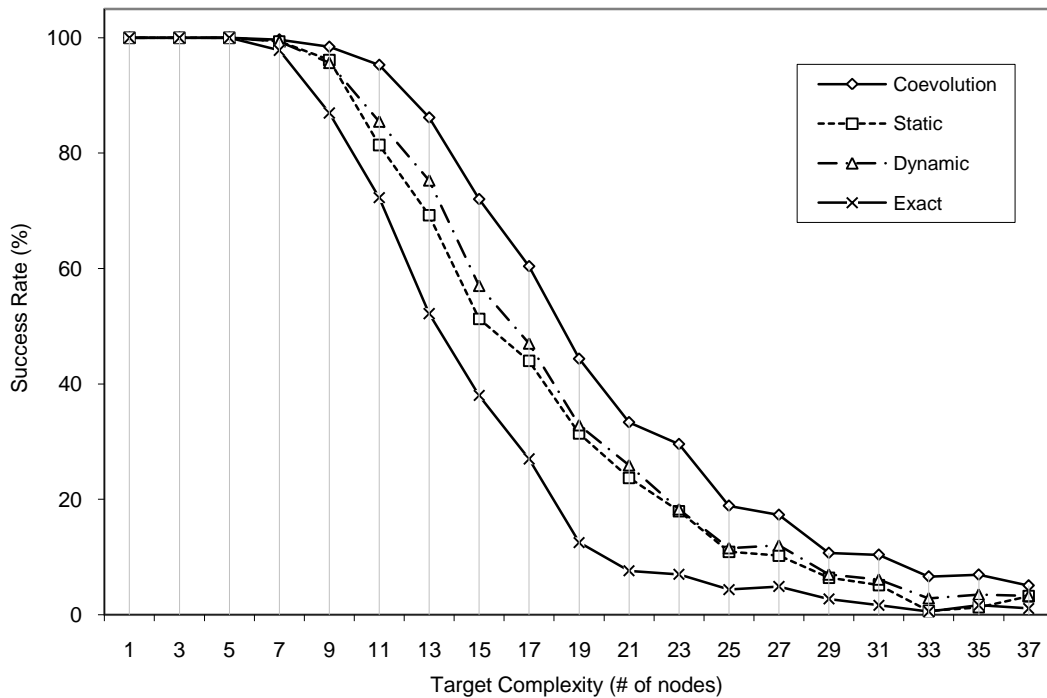
**Figure 4.10. The Chi-square p-values for significance of convergence versus complexity between the coevolution algorithm and each compared algorithm.**

The four fitness algorithms described in the first experiment were tested on the randomly generated target symbolic functions. For each run, all algorithms were initialized with the same initial populations and control parameters. We used the same experimental setup and controls as in the previous experiments.

Each run is stopped after 10 million function evaluations. Then the best individual is tested for a perfect fit to the test data, and a tally of the successful convergences is recorded for each complexity. The percent of successful convergences versus complexity for each alternative algorithm is plotted in Figure 4.11.

We have performed a Chi-Square statistical test between coevolution and each algorithm. The difference in convergence is found to be statistically significant ( $p < 0.05$ ) for all complexities between 9 and 37. More samples at higher complexities are needed to conclude the significance at 37.

A Chi-Square p-value  $< 0.05$  is shown to indicate statistical significance. At low complexities, all algorithms have 100% convergence and have no statistical difference. The p-values for higher complexities show that coevolution has statistically significant higher convergence than the other three algorithms compared. More samples are needed to show significance at complexities 37 and higher.



**Figure 4.11.** The percent of successful convergence after 10 million point evaluations versus the target function complexity (the number of nodes in the binary expression tree).

We see that all algorithms have a very high probability of success for simple functions. Furthermore, all algorithms experience a drop in success with an increase in the complexity of the function, but at different rates.

The coevolution algorithm has the highest success rate in general. It maintains a 5-10% higher convergence rate over the other fitness algorithms involving the 11 to 27 complexity functions. Most notably, coevolution maintains a 1-4% advantage over the 29 to 37 complexities where the other algorithms have 0-3% successful convergence overall.

The static and dynamic fitness approximation algorithms perform significantly better in comparison to the exact fitness algorithm with the 9 to 37 complexity functions. In

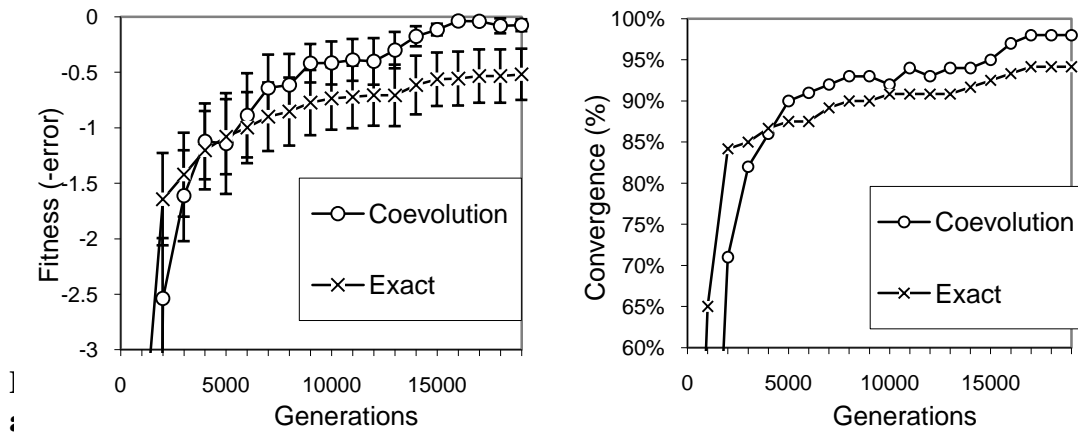
the previous experiment, we saw that the exact fitness algorithm achieves higher fitness values, but here we are only measuring convergence, and the fitness prediction algorithms converge significantly more on average over random functions. The exact fitness algorithm achieves many fewer generations for the same number of point evaluations and may simply be lacking some amount of exploration from crossovers and mutations to converge on the final solution.

Next we look at the improvement factor in order to compare coevolution pair-wise with the other three approaches. The improvement factor is the ratio of convergence of coevolution to the compared algorithm, over complexity:

$$\text{Improvement Factor} = \frac{\% \text{ convergence of coevolution}}{\% \text{ convergence of compared algorithm}}$$

An improvement factor of one indicates the two algorithms have the same performance. A factor of less than one indicates that coevolution performed worse. Greater than one indicates coevolution performed better. For example, a factor of two indicates coevolution had twice the convergence at a given complexity.

Though all algorithms decrease in convergence with increasing complexity functions (Figure 4.11), the improvement factor for coevolution tends to increase (Figure 4.12). Statistical testing (Figure 4.10 and Table 4.4) demonstrates this growth as significant for complexities 11 and higher. Based on this observation we conclude that coevolution may offer greater tolerance to growing complexity.



**Figure 4.13. Fitness and percent of runs converged versus generations throughout evolution on the function  $f_2(x)$ . Error bars show the standard error. Note that exact evaluations are performing significantly more computational effort per generation.**

### Improving Solution Reliability

One important effect of fitness prediction is the adaptation of fitness pressures, which causes the evolutionary focus to change throughout evolution. In this section, we examine how this effect impacts the solutions found by comparing performance by generation, rather than computational effort. We also examine the difference in solution bloat when using coevolved fitness predictors.

### Comparing Performance by Generation

We measure the fitness and convergence of 100 runs versus the number of generations (not point evaluations as before). Note that in our previous experiments, coevolution achieves many more generations with the same number of point evaluations (computational effort) by utilizing the fitness predictor.

The experiment is identical to the previous experiments; however, we run the exact-fitness algorithm out to billions of point evaluations so that we can compare performance based on the number of generations rather than the amount of

computational effort.

Figure 4.13 shows the performance of each algorithm over 20,000 generations while regressing  $f_2(x)$ . This is sufficiently long enough for both algorithms to achieve 90% convergence or higher.

The exact fitness algorithm starts with a clear lead over coevolution in both fitness and convergence in early generations. However, at approximately 4000 generations coevolution begins to dominate the exact fitness algorithm over the averaged 100 test runs.

This empirical result on  $f_2(x)$  suggests that coevolution outperforms the use of exact fitness measurements even when ignoring the high cost of exact fitness values. There are several possible explanations for this. Fitness approximation can drive solutions to unexplored areas of the domain (Booker, Dennis et al. 1999; Regis and Shoemaker 2005), perhaps increasing convergence. Additionally, adapting the fitness approximation can destabilize local optima solutions, as also noted by (Pagie and Hogeweg 1997; Jin 2005). When individuals converge to local optima in the fitness predictor, predictors react to approximate the region more accurately. The better the local optima solutions are, the more stable they will be during the predictor transition. Since the predictions shift data point emphasis, the improvement may also be related techniques such as boosting or adaptive weighting. Although this behavior may be an important advantage of coevolved predictors, understanding it is beyond the scope of this chapter.

### ***Reducing Bloat***

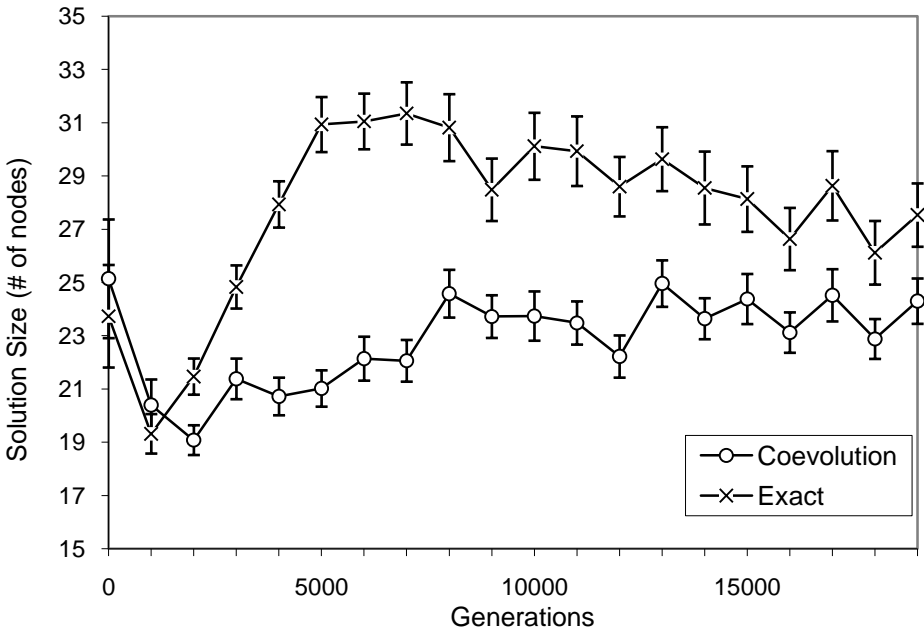
A challenging problem in many genetic programming domains is dealing with bloat. Bloated solutions are those that are excessively complicated. In relation to machine

learning, bloat can be thought of as “overfitting”, in which solutions evolve complex structures that do not exist in the real system.

Bloat can also be problematic in symbolic regression. Figure 4.14 shows the size of the best solution during evolution on  $f_2(x)$  averaged over 100 test runs. Function  $f_2(x)$  is a very simple nonlinear target function that has two difficult local optima. This is a good first example because the local optima may be cause for extra bloat during evolution. Later we compare bloat on randomly generated functions.

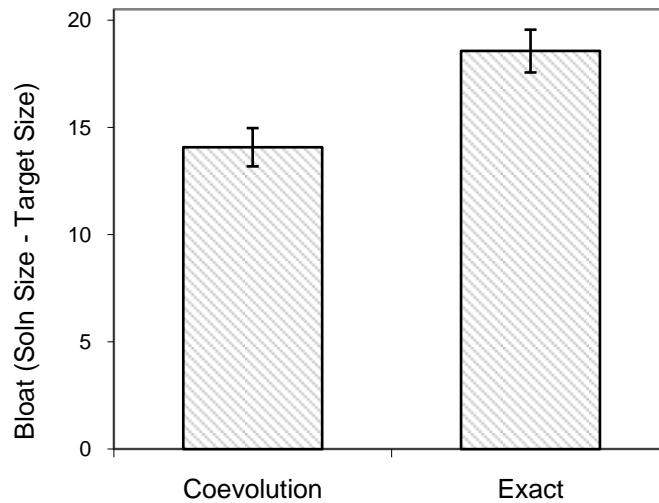
In this instance, size, defined as the number of nodes on the binary tree, is synonymous with the complexity metric used earlier.

On average, coevolution maintains significantly less complex solutions during evolution than the algorithm using exact fitness calculations. The exact fitness solutions balloon near 5000 generations while coevolution experiences solution sizes



**Figure 4.14.** The size of the best solution during evolution of  $f_2(x)$  averaged over 100 test runs. Error bars show the standard error.





**Figure 4.15. The bloat of final converged solutions averaged over 500 randomly generated target functions. Error bars show the standard error.**

that are both lower and more consistent.

This preliminary result from  $f_2(x)$  suggest fitness prediction is less susceptible to bloat. To get an idea if this could be a general trend, we compared solution sizes of both algorithms on randomly generated target functions where both algorithms are allowed to fully converge.

Figure 4.15 shows the bloat of final solutions of both algorithms on 500 randomly generated target functions. Coevolution yields less bloated solutions on average for randomly generated functions as well. Here we define bloat as the solution size minus the target function size. Each algorithm is tested on the same target functions and only target functions in which both algorithms converged are considered. Note that bloat reduction can also improve computational performance per point evaluation, since smaller expressions can be evaluated faster.

Coevolutionary bloat reduction is an important observation for this chapter, but deeper

analysis is beyond the current scope. One hypothesis is that the fitness landscape imposed by fitness prediction is simpler and therefore inherently biased towards simpler solutions. In the case of a subset predictor as used here, the sample is less likely to encompass fine detail in training data features, thereby reducing pressure to explain detail or noise features until the solutions have converged on the larger trends first. However, we leave deeper analysis to future work.

## **Conclusions**

This chapter proposed a coevolution algorithm to address three fundamental challenges faced when using fitness modeling in evolutionary algorithms: (1) the model training investment, (2) choosing a level of approximation, and (3) loss of accuracy. The coevolutionary framework uses three populations: Problem solutions, fitness predictors, and fitness trainers. Solutions evolve to maximize their predicted fitness, fitness trainers are selected to cause the most inconsistencies between fitness predictors, and finally fitness predictors evolve to minimize error in predicting the fitness trainers.

For the problem of symbolic regression, we have shown the following advantages:

- 4. Computational performance improvement:** Coevolution provides substantial performance improvement over exact fitness, random sample, and dynamic sample fitness algorithms. On simple manually designed test problems, coevolution achieves higher average fitness values and more reliable convergence with significantly less computational effort in each case. Coevolution also performs competitively with other recently published symbolic regression methods. In these experiments, coevolution achieves significantly higher convergence on challenging experiments such as

trigonometric derivations and has a similar performance on simple experiments such as polynomial targets.

5. **Scaling:** In experimentation on randomly generated benchmarks, coevolution shows higher performance over all solution complexities tested. The factor of improvement increases as complexity rises.
6. **Performance by generation:** Empirical results show that coevolving fitness predictors can yield higher fitness solutions compared to the exact fitness algorithm even when disregarding savings in computational effort. This suggests that the transformation of the fitness landscape is in itself beneficial.
7. **Bloat reduction:** Empirical results suggest that, on average, coevolution yields less bloated solutions for randomly generated target functions.

Finally, fitness prediction is a technique that can be applied in many domains and general problems. Certain problems that have traditionally been poorly suited for fitness approximation (e.g. symbolic regression) or coevolution could benefit from this coevolutionary approach – such as increasing computational performance, scaling to higher complexity problems, improving convergence, and reducing bloat.

## CHAPTER 5. RANK PREDICTION

### **Summary**

Many applications of evolutionary algorithms utilize fitness approximations, for example coarse-grained simulations in lieu of computationally intensive simulations. Here, we propose that it is better to learn approximations that accurately predict the *ranks* of individuals rather than explicitly estimating their real-valued fitness values. We present an algorithm that coevolves a *rank-predictor* which optimizes to accurately rank the evolving solution population. We compare this method with a similar algorithm that uses *fitness-predictors* to approximate real-valued fitness values. We benchmark the two approaches using thousands of randomly-generated test problems in Symbolic Regression with varying difficulties. The rank prediction method showed a 5-fold reduction in computational effort for similar convergence rates. Rank prediction also produced less bloated solutions than fitness prediction.

### **Introduction**

In practice, many applications of evolutionary computation involve expensive fitness calculations (Jin, Olhofer et al. 2001; Ong, Nair et al. 2003). For example, some problems involve simulating the performance of evolved robotics or structures. Others commonly involve evaluating a solution over a large dataset.

One method to address the computational difficulty of fitness calculation is fitness modeling and approximation (Jin 2005). Fitness models are often coarse approximations of the full fitness calculation – for example, a coarse simulation, or subset of the dataset – chosen ahead of time to replace the full fitness function.

One general method to improve performance using fitness approximations in arbitrary applications is the Coevolution of Fitness Predictors algorithm (Schmidt and Lipson

2006; Schmidt and Lipson 2006; Schmidt and Lipson 2008). Here, the concept of a fitness predictor is to estimate the exact fitness value of an individual with an extremely coarse and light-weight approximation. Instead of specifying the approximation ahead of time, fitness predictors are coevolved, optimizing their ability to estimate the exact fitness values of the current solution population.

A surprising result from this method is that it can improve performance even with extremely coarse fitness approximations. For example, in the symbolic regression (Koza 1992) problem, the fitness predictors can maintain an objective fitness gradient by evaluating solutions on as few as four data points in a data set of thousands of points and tens of variables (Schmidt and Lipson 2007). In such extreme cases, the fitness approximations are almost certainly inaccurate, but still allow evolutionary progress on the objective fitness.

In this chapter, we propose that the primary mechanism by which fitness approximations improve performance is by providing accurate rankings of individuals, rather than accurate fitness values as originally intended. Furthermore, we suggest that performance can be improved even further by selecting approximations that are optimized to rank solutions, rather than model their fitness directly.

To test this idea, we use two implementations of the Coevolution of Fitness Predictors algorithm for symbolic regression (Schmidt and Lipson 2008). The first is the standard fitness predictor algorithm which coevolves a small subset of the total training data to measure error. The second is identical, however fitness predictors are replaced with rank predictors, which are optimized to rank solutions, rather than model their fitness values. We then test the performance of these two algorithms on thousands of generated test problems and observe their differences over varying problem

difficulties.

In the remaining sections, we describe related work in fitness approximation and introduce the basic algorithm for coevolving fitness or rank predictors. We then detail our experiments and test problems on the symbolic regression problem and present results. Finally, we conclude with discussion and final remarks.

## **Background**

Mentioned above, fitness modeling is the technique of using a predefined model or coarse simulation to approximate the fitness calculation in evolutionary algorithms; especially in cases where the exact fitness requires an expensive simulation or physical experiment. In contrast, fitness predictors are a type of fitness model that is so coarse that they cannot approximate the entire fitness landscape. Instead fitness predictors must be adapted throughout evolution.

In this chapter, we use a sub-sampling of training data for the predictor structure. For the fitness predictor, the sub-sample is optimized to match the fitness of the entire data set, while the rank predictor simply picks points that accurately rank the solutions. In both cases, the sample is optimized in a second coevolving population (Schmidt and Lipson 2008).

## **Algorithm**

### ***Fitness and Rank Predictors***

The objective of a fitness predictor is to approximate the expensive, exact fitness calculation of an evolving problem solution. The objective of a rank predictor however is to provide a ranking of solutions that corresponds to their ranking based on their exact fitness values.

These two types can be very similar in implementation. A fitness predictor does in fact produce a ranking of solutions – a ranking based on the predicted fitness values.

In fact, in our implementation, we represent rank and fitness predictors identically. The primary difference is the objective they are optimized for: producing an accurate ranking or a representative fitness value.

Both rank and fitness predictors produce a numeric value. For the fitness predictor, this value is optimized to match the exact fitness value of the solutions in the current population. The numeric value produced by the rank predictors has no discernable scale or magnitude; it is simply a value that is likely correlated with the exact fitness. Furthermore, the rank predictors are optimized such that if this value is used to rank the solution population, it produces a similar ranking to that based on the exact fitness values.

In our experiments, we compare the two methods on the symbolic regression problem where fitness is measured by error on a dataset. Here the fitness and rank predictors are encoded as a small subset of the total training data. The subset indicates to evaluate the solutions and measure error only on these data points. We used a fixed subset size of 16, where the total training data set size is 500.

### ***Fitness and Rank Trainers***

Because fitness and rank predictors are very coarse approximations, they need to be optimized to approximate for the current solution population. Therefore, we need to calculate the exact fitness (error on all data points) of some solutions from the current generation in order to train the predictors. These example solutions are known as fitness trainers.

Fitness trainers are selected in order to help predictors optimize to the current solutions. To do this, the algorithm chooses a solution whose predicted rank or fitness has the least confidence. For example, we select the solution with the highest variance (Bongard and Lipson 2005; Jin and Branke 2005) in predicted fitness, or highest variance in predicted rank, among the current rank or fitness predictors.

Additionally, old trainers are discarded to keep the predictors optimizing to only recent solutions. If the population diverges away from older solutions, we don't want to optimize the predictors on those solutions any longer. In our experiments, we discard trainers older than 1000 generations.

The population of trainers allows us to define a fitness, or optimization criterion, for the predictors. In the case of fitness predictors, where  $i$  spans the set of trainers, this metric is:

$$-\frac{1}{N} \sum_i |fitness(i) - prediction(i)|$$

Very simply, this rewards the fitness predictors to accurately reproduce the exact fitness value.

In the case of the rank predictor, where  $i$  and  $j$  span the set of pairs of trainers, the metric is:

$$-\frac{1}{N^2} \sum_{i,j \text{ pairs}} \begin{cases} 0 & i \text{ and } j \text{ ordered correctly} \\ 1 & \text{otherwise} \end{cases}$$

This rewards rank predictors for correctly ordering pairs of solutions – or equivalently, correctly ranking all trainers.



### ***Coevolution Algorithm***

The coevolution algorithm (Schmidt and Lipson 2008) that we modify in this chapter has three populations: Problem solutions, fitness predictors, and fitness trainers. As described earlier, fitness trainers are a set of solutions chosen to train the fitness and rank predictors on.

The algorithm chooses individuals from the solution population to calculate exact fitness values in order to train the fitness or rank predictors. The algorithm then evolves the solution population using the highest ranked fitness or rank predictor, and evolves the predictors using the fitness trainers.

### **Experimental Setup**

In this section we detail our experimental methods to test the impact of using rank predictions rather than fitness approximations. We perform identical experiments on two algorithms: (1) the coevolved rank predictor algorithm, and (2) the coevolved fitness predictor algorithm (Schmidt and Lipson 2008).

We experiment on the Symbolic Regression problem because it is a ubiquitous and important problem in genetic programming (Koza 1992). Additionally, we can easily vary the problem complexity and the problem dimensionality.

Symbolic regression (Koza 1992) is the problem of identifying the simplest equation (Grünwald 2000) that most accurately fits a given set of data. Symbolic regression has a wide range of applications, such as prediction, classification, modeling, and system identification.

Recently, symbolic regression has been used to detect conserved quantities data representing physical laws of nature (Schmidt and Lipson 2009), infer the differential

equations in dynamical systems (Bongard and Lipson 2007).

### ***Symbolic Regression***

See the description in the section "Symbolic Regression," on page 4.

### ***Test Problems***

We measured performance of each algorithm on randomly generated test problems. To generate a random problem in symbolic regression, we simply need a random target equation to find and a set of data corresponding to that equation for the fitness error metric.

We experiment varying two characteristics of the random symbolic regression problems: (1) the dimensionality of the data (i.e. the number of variables in the data set), and (2) the complexity of the target function (i.e. the size of the equation's binary parse tree). Both of these characteristics factor into the problem's difficulty. Increasing dimensionality increases the base set of possible variables for the equation may use, while increasing complexity increases the chances of couples nonlinear features.

The first step in our random test problem generation is to randomly sample the dimensionality of the problem. We pick a random number of variables between one and ten.

Next, we generate a random equation which can use any of these variables. We generate a random equation in the same fashion that we generate random individuals in the evolutionary algorithm.

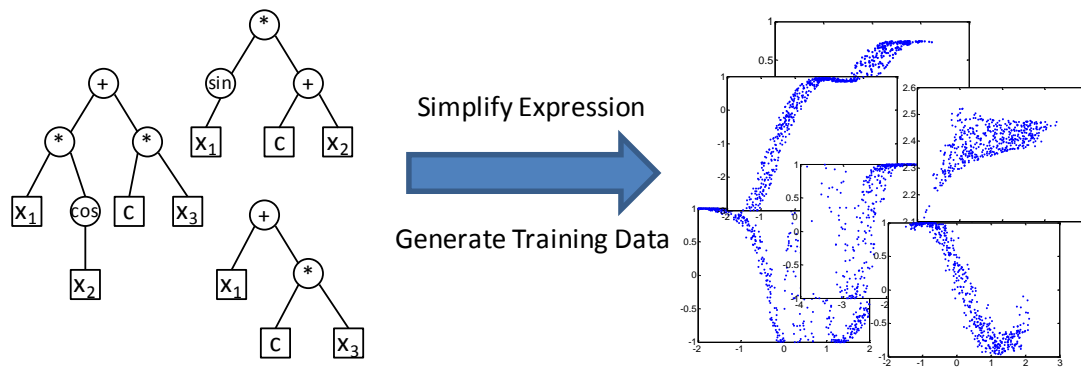
Many randomly generated equations may have compressible terms. For example,  $f(x) = 4.211 + 0.93 x^2 + 1.23$  is equivalent to  $f(x) = 0.93 x^2 + 5.441$ . Therefore, we perform a symbolic simplification on the randomly generated equation in order to get an

accurate measure of the target equations complexity. We measure complexity of the problem as the total nodes in the binary tree representation of the equation. For example, the complexity of the equation just above is 5.

We repeat this step as necessary in order to get a uniform distribution of problem complexities. We continue generating and simplifying equations in order to uniformly sweep the problem complexities between 1 and 32.

Next, we randomly sample the input values of the equation 500 times to create a dataset. These variables are sampled from a normal distribution around the origin, with standard deviation of two. The equation is then evaluated on these variables in order to get the target output value. Several examples of training data are shown in Figure 5.1.

Finally, we also generate a separate validation data set of 500 points. The validation data set is created in the same fashion as the training data set, however the input



**Figure 5.1.** The generation of random test problems for symbolic regression. We start by picking a random number of inputs, between one and ten. We then generate a random equation using these inputs and simplify the equation before measuring its complexity (the number of nodes in the binary tree). We then generate a random training data set by sampling the input variables around the origin and evaluating the target equation on these data points. We then generate a validation data set in a similar fashion, but with a wider range around the origin to test if the solutions extrapolate to the exact solution.

variables are sampled with a standard deviation of three. By using a broader input sampling, we can use the validation dataset to test whether solutions extrapolate in their predictions to unseen data.

We also use this to measure the percent of times the algorithms find the exact solution – if the algorithm achieves near zero error on the extrapolated validation dataset. Since we are not adding any noise to the dataset, we expect the algorithms to reach zero error on the generated data, if the exact solution is in fact found.

### ***Measuring Performance***

We tested each algorithm on 1000 randomly generated symbolic regression problems. Each evolutionary search was performed on a single quad core computer.

Evolution was stopped if the algorithm identified a zero error solution on the validation data set (i.e. less than  $10^{-3}$  normalized mean absolute error), or when the algorithm reached one million generations.

Throughout each search, we log the best equation, its fitness (i.e. normalized mean absolute error) on the training and validation sets, its complexity, and the total computational effort. We measure computational effort as the total equation evaluations performed in fitness calculations.

The fitness of the normalized mean absolute error is normalized using the standard deviation of the target output values. The normalized fitness allows comparing fitness values between evolution runs and detecting convergence to the exact target solution more easily. In all figures, we show the fitness on the validation data set (i.e. the normalized mean absolute error on the validation data).

### ***Algorithm Settings***

We use the symbolic regression algorithm described in (Schmidt and Lipson 2008) as the basis for our implementation. We simply swap out the fitness criterion for the fitness predictor for the rank predictor criterion, described earlier.

We use deterministic crowding selection (Mahfoud 1995), with 1% mutation probability and 75% crossover probability. The encoding is an acyclic graph of 64 operations/nodes (Schmidt and Lipson 2007). The operation set allowed addition, subtraction, multiply, divide, sine, and cosine operations.

### **Experimental Results**

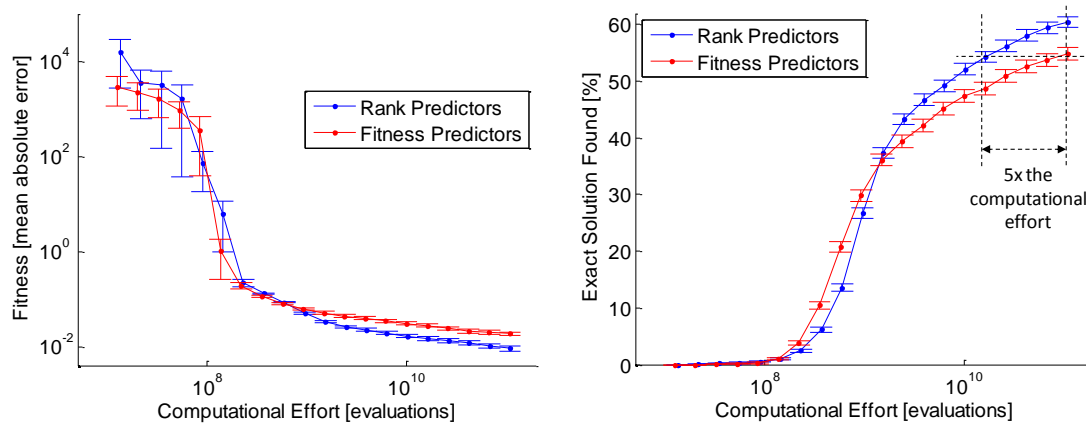
This section summarizes the experimental results comparing the two algorithms: (1) the standard fitness prediction algorithm, and (2) the rank predictor algorithm.

#### ***Fitness and Convergence***

We first observe the fitness of each algorithm over the course of the evolutionary search, with the time measured in computational effort – the total point evaluations of all equations in fitness calculations, predictions, or rank predictions.

The fitness values plotted in Figure 5.2 show both algorithms have similar trends on the randomly generated test problems, suggesting that the algorithms experience similar optima during their searches. Despite this, we see a clear difference in the fitness performance over time, with rank predictors achieving lower error.

This may also reflect the difference in convergences to the exact problem solution, also plotted in Figure 5.2. Here we notice that the fitness predictor algorithm begins finding exact solutions slightly sooner than the rank predictor algorithm. However, it is quickly overcome by the rank predictor algorithm.



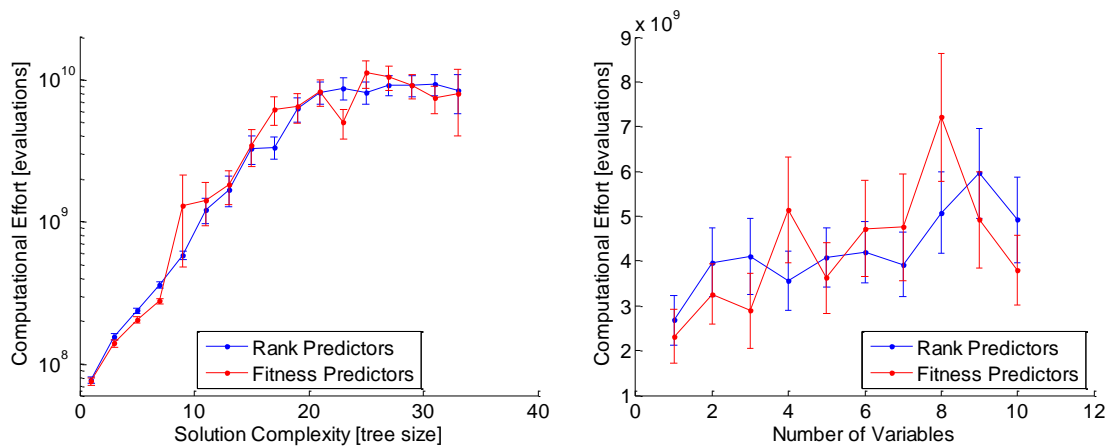
**Figure 5.2.** The fitness and convergence rate to the exact solution of each algorithm versus the total computational effort of each trial. The fitness (left) is the normalized mean absolute error on the validation data set. Convergence to the exact solution (right) represents the percent of the trials that identify solutions that have less than epsilon error on the validation data set. Error bars indicate the standard error. The performance of the algorithm without using prediction at all is several order of magnitude higher in computational effort and is not shown.

Later in the evolutionary searches, the rank predictor algorithm shows a clear trend of finding the exact problem solution more often – reaching 55% average convergence rate in less than 1/5 the time than that of the fitness predictor method.

### *Computational Effort*

We also compared the total computational effort each algorithm required to find the exact problem solution – in cases where the algorithm did indeed find the exact solution. Here, we looked at the computational effort versus the complexity of the target solution and the dimensionality of the datasets for each evolutionary search.

In response to increasing target solution complexity, shown in Figure 5.3, both algorithms show very similar trends. We do see a small difference, where the fitness predictor algorithm tended to find the exact solution slightly faster for the simplest problems. At higher complexities, the difference is less noticeable, however rank



**Figure 5.3.** The computational effort required when the exact solution was found versus the target equation complexity (left) and the number of variables in the dataset (right). Each algorithm found the exact solution with different frequencies; these plots show the computation effort for when the algorithms did find the exact solution. The error bars indicate the standard error.

predictors tended to require slightly less effort at most higher complexities than the fitness predictors.

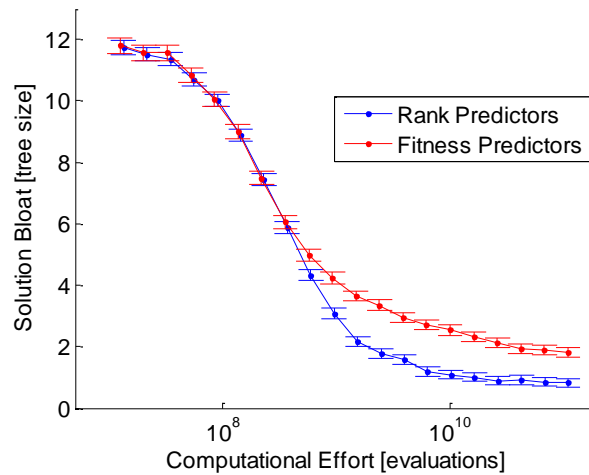
There is a similar trend found in the computational effort to find the exact solution versus the number of variables in the problem datasets (Figure 5.3). Computational effort tended to increase with dimensionality for both algorithms. Again, fitness predictors tended to require slightly less effort on average for the lower dimensions.

### ***Solution Bloat***

Finally, we looked at the solution bloat that both algorithms experienced over the course of the evolutionary searches.

We define bloat as the difference between the binary tree size of the best solution in the population and the target solution. Therefore, the bloated solutions have positive bloat values, and underfit solutions have negative bloat values.

The bloat results (Figure 5.4) show that both algorithms begin with highly bloated



**Figure 5.4. The mean solution bloat of the best solution versus the computational effort. Solution bloat is defined as the binary tree size of the best individual in the population minus the size of the target solution. Error bars indicate the standard error.**

solutions, which decrease over the search toward the target solution on average.

Interestingly, fitness predictors are slightly more bloated on average than the rank predictors. This is only true however later during the evolutionary searches. However, it's unclear if this is due to the lesser convergence of the fitness predictors.

## Discussion

The results in the previous sections show several interesting trends which highlight the difference between the two algorithms.

Most significantly, we found that the rank predictor algorithm found the exact solution more often on the hardest problems which took the most computational effort to solve. The rank predictor algorithm also found solutions with higher objective fitness on average, despite only being evolved to only improve the solutions' ranks.

Overall, results in computational effort, for both the test problem complexity and the



number of variables in the dataset, were similar. This suggests that there was not great difference in speed to find the exact solution between the two algorithms – when it is indeed found. Instead, the benefit must be arising from finding the exact problem solution more often.

Interestingly, the fitness predictor algorithm achieved slightly higher performance than the rank predictors early in the evolutionary searches, and for the simpler test problems. Additionally, the fitness predictor algorithm experienced more bloat on average than the rank predictor algorithm. This suggests that fitness predictors may be placing stronger pressure to fit detailed features in the data set. In simple test problems, this may boost convergence to the exact solution. In more difficult problems however, it could result in excessive bloat.

This may be the primary reason rank predictors outperformed the fitness predictors. By optimizing solution ranking, rather than explicit fitness values, they may not need to emphasize large errors or detailed features to create accurate fitness values. They only need to emphasize the points of disagreement between solutions in order to find an effective ranking.

## **Conclusion**

In summary, many applications in evolutionary computation rely on fitness approximation and modeling. Instead of using fitness models which approximate the absolute fitness value, we proposed optimizing rank predictors – approximations which can accurately rank solutions in correspondence with the absolute fitness.

We compared the difference between optimizing modeled fitness values and optimizing solution rankings using a coevolutionary algorithm which optimizes either fitness predictors or rank predictors with the evolving problem solutions. We tested

both methods on the symbolic regression problem using thousands of test problems, varying in problem complexity and number of variables.

Our results found rank predictors strongly outperform fitness predictors, achieving higher fitness on average and identifying the exact problem solution more often. Interestingly, when solutions are found by both algorithms, both algorithms used similar amounts of computational effort to find solutions, suggesting the primary benefit from rank prediction comes from identifying the exact solution more often (i.e. more reliably).

## CHAPTER 6. META-OBJECTIVES IN EVOLUTIONARY SEARCH

### **Summary**

In this chapter, we explore the impact of meta-objectives – optimizing secondary objectives – in an evolutionary search. Ordinarily, evolutionary algorithms attempt to optimize a primary objective, such as minimizing error. Here, we consider three other secondary objectives: genotypic age, genotypic novelty, and solution complexity. Recent research has shown each of these traits to be important in evolutionary search individually. Here, we examine the impact of optimizing all combinations of these objectives simultaneously, to improve the original primary objective, in an explicit multi-objective search. We first compare an explicit multi-objective algorithm that optimizes error and age objectives with the existing single-objective age algorithm on the Symbolic Regression problem. Results show that the multi-objective approach identifies the exact target solution more often than the age-layered population and standard population methods. The multi-objective method also performs better on higher complexity problems and higher dimensional datasets – finding global optima with less computational effort. Next, we repeated this experiment for each combination of the four objectives. Results show that age yields the greatest improvement in performance for a single extra objective. Performance improves even more when additionally optimizing for age and novelty. Optimizing for complexity tended to only improve the Error-Complexity Pareto volume performance.

### **Introduction**

A common problem in many applications of evolutionary algorithms is when the progress of the algorithm stagnates and solutions stop improving. Expending additional computational effort in the evolution often fails to make any substantial progress. This problem is known as *premature convergence* (Kenneth Alan De 1975;

Louis and Rawlins 1992; Conor 1996).

A common method for dealing with premature convergence is to perform many evolutionary searches, randomizing and restarting the search multiple times (Jansen 2002; Auger and Hansen 2005). This approach can be wasteful however, as the entire population is repeatedly thrown out. There is also the difficulty of deciding when to restart, and the possibility that the converged population could continue improving with additional diversity.

One of the best performing methods in the genetic programming literature for addressing premature convergence is the Age-Layered Population Structure (ALPS) method (Hornby 2006; Hornby 2009). ALPS uses a special notion of *age* – how long genotypic material has existed in the population – in order to partition the evolving population into age layers (see Figure 6.2). The algorithm adds new random individuals into the youngest population layer throughout the search, and layers evolve independently of others. As a result, the youngest layers, do not immediately compete with the oldest and most fit solutions. Implementation of the ALPS algorithm, however, requires new parameters, such as how to pick age layer cutoffs and how many solutions to keep in each layer, etc.

The concept of age in the ALPS algorithm is an example of a secondary objective. The ALPS algorithm uses this objective to partition the population to significantly improve search performance (Hornby 2006; Hornby 2009).

In this chapter, we first consider using the ALPS concept of age as a fundamental property in the evolutionary optimization. Rather than using age to partition the population into layers, we use age as an independent dimension in a multi-objective Pareto front optimization. In this context, a solution is selected for if it has both higher

fitness and lower genotypic age than other solutions.

A completely multi-objective approach allows us to consider adding other secondary objectives. Our hypothesis is that, based on the impact of age, other seemingly unrelated objects may further improve performance.

We consider explicitly optimizing two other objectives in addition to age: solution complexity and genotypic novelty. We test the impact that optimizing all possible combinations of these objectives has on the overall performance on the primary objective.

### **Heuristics**

Here we introduce the secondary objective metrics. In all experiments we use a primary objective (minimize error), with zero or more secondary objectives.

### ***Complexity***

Complexity is a commonly used secondary objective in genetic programming (Mark, Guido et al. 2007; Schmidt and Lipson 2009). Complexity generally measures the size or content of a solution. Many algorithms explicitly minimize, or penalize for complexity in order to reduce bloat (Banzhaf and Langdon 2002) – the tendency to evolve exceedingly complex solutions.

Often, complexity is incorporated as a penalty in the primary fitness objective when solutions become large enough. This effectively establishes a fixed tradeoff between complexity and fitness. When used in multi-objective optimization instead, the complexity metric biases the search toward simpler solutions (Edwin and Jordan 2003). Simple solutions are favored because they are non-dominated in age.

In our experiments in symbolic regression, we measure complexity as the tree size –

the number nodes in the expression's binary tree representation.

### *Age*

Interestingly, the concept of genotypic age as used in ALPS has shown to be one of the best approaches for avoiding premature convergence and improving results (Hornby 2009). Our goal in this chapter is to develop this idea further by utilizing genotypic age as a fundamental search trait.

The age of a solution is generally measured in generations, or alternatively computational effort measured in fitness evaluations for steady-state algorithms (Hornby 2009).

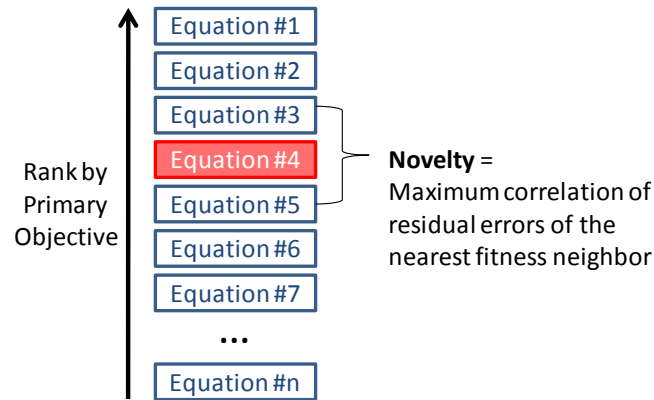
All randomly initialized individuals start with age of one. With each successive application of a variation operator, the age of an individual is incremented by one. This alone measures the amount of time an individual has existed in the population. However, we are more interested in the age of the genotype.

To measure the age of the genotype, we need to pass on ages during crossover and mutation events. There are several options, such as taking the age of the most similar parent, taking the average age of the parents, etc. The best method reported in the literature (Hornby 2009), and the method we use, is to inherit the maximum age of the parents.

Therefore, the age is a measure of how long the oldest part of the genotype has existed in population.

### *Novelty*

Novelty is a measure of how new or original a solution is, or how densely the search has explored on similar genotypes. It has also been suggested as a primary search



**Figure 6.1. The novelty objective of a solution. Here, the novelty of equation #4 is equal to the maximum correlation of its residual errors with its two nearest neighbors in terms of fitness.**

objective (Lehman and Stanley 2010), where the population is evolved in order to maximize novelty. Maximizing novelty has the effect of increasing the search coverage, ensuring a high degree of exploration – or even a maximum amount of exploration versus the computational effort.

Novelty can also be thought of as a diversity metric. The higher novelty values in a population, the greater the diversity. Therefore, novelty will also prevent pre-mature convergence, but in a more direct way than age.

In our experiments, we measure novelty as the correlation of a solution with other solutions of similar fitness. The higher the correlation, the less novel the solution is. We first sort all solutions by their fitness (the primary objective, such as error on a data set). We then calculate the correlation coefficients of each solution with its closest fitness neighbor. We then define novelty measure as one minus this correlation value.

### ***Random Objectives***

In our experiments, we also use random objectives in order to more accurately

measure the impact of each secondary objective. If a particular combination does use one of the three secondary objectives, the objective is replaced with a random objective.

Each new solution is given a random score on each random objective when initialized. The random scores are inherited during crossover. This allows some solutions to be non-dominated by chance, but otherwise implies no other metric of the solution.

In effect, this allows us to measure the impact of each objective over a random noise objective, since an algorithm may otherwise have sensitivity to the dimensionality of the multi-objective optimization.

## **Algorithm**

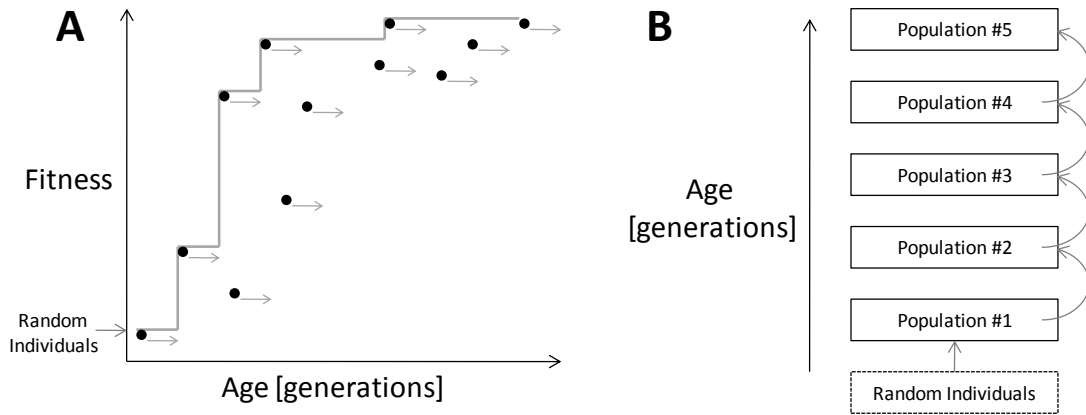
### ***Age-Fitness Algorithm***

As in the ALPS method, random individuals are added into the population at each generation. Rather than flowing up the age layers, they flow through a two-dimensional space of fitness and age (see Figure 6.2). Young solutions exist in the same population as the oldest and most fit, but persist because they are non-dominated on the age dimension of the Pareto space.

A key benefit of the proposed approach is that it does not require a population partitioning or structuring. For example it does not constrain intermediate layer sizes, the number of total layers, or layer partitions. These variations all exist within the larger Pareto space of the search, allowing the age-fitness distributions to vary dynamically.

Like ALPS, this approach makes no assumptions about the underlying solution representation. Therefore, it can be applied to nearly any evolutionary search problem





**Figure 6.2.** The Age-Fitness Pareto Population algorithm (A) considers a single population of individuals moving in a two-dimensional Age-Fitness Pareto space. Individuals are selected for if they simultaneous have higher fitness values and lower age than other individuals. Ages increase every generation, or are inherited during crossover, and new random individuals are added with zero age. In the Age-Layered Population Structure (ALPS) algorithm, there are several layers of populations for each age group. New individuals are injected to the youngest population, and individuals migrate to older populations as their age increases.

to improve the optimization performance.

### *Multi-objective Optimization*

There are a number of ways to implement multi-objective evolution (Ekárt and Németh 2001; Kalyanmoy and Deb 2001; Zhang and Rockett 2007). In this chapter, we use the simple random mating with tournament selection method.

Each generation, we select random pairs of individuals, cross and mutate them probabilistically, and add them to current population. Additionally, a new random individual is added to the population each generation.

We specify a target population size – analogous to the population size in a traditional evolutionary algorithm. The goal of the selection is to remove dominated individuals from the population until the target population size is reached.

We used the SPEA2 (Strength Pareto Evolutionary Algorithm) for selection (Zitzler, Laumanns et al. 2001). SPEA2 is one of the most popular multi-objective methods. It scores and selects solutions based on how many other solutions dominate it. Non-dominated solutions on the Pareto frontier are always selected. If the number of solutions on the Pareto frontier are larger than the target population size, SPEA2 iteratively removes the solution with the closest neighbors.

## **Experiments**

We compare several combinations of ALPS and multiple objectives on the Symbolic Regression problem. Here we describe the experimental setup.

### ***Symbolic regression***

See the description in the section "Symbolic Regression" on page 4.

### ***Random test problems***

We tested each algorithm on 1000 randomly symbolic regression problems. Each evolutionary search was performed on a single quad-core computer. The testing procedure was the same as described in the section "Test Problems" on page 54.

### ***Algorithm Settings***

We used standard algorithm settings for symbolic regression – 75% crossover, 1% mutation. We used a population size of 1000. This was large enough such that the Pareto frontier always fit inside the population in all experiments. Solutions were allowed to use add, subtract, multiply, divide, sine, cosine, a variable, or a constant coefficient.

## **Results**

Results are split into sections: the age-fitness optimization algorithm, and the

combinations of multiple objectives.

### *Age and Fitness Objectives*

This section summarizes the experimental results comparing the three algorithms: (1) the ALPS algorithm, (2) Age-Fitness Pareto algorithm, and (3) the Deterministic Crowding algorithm with randomized individuals.

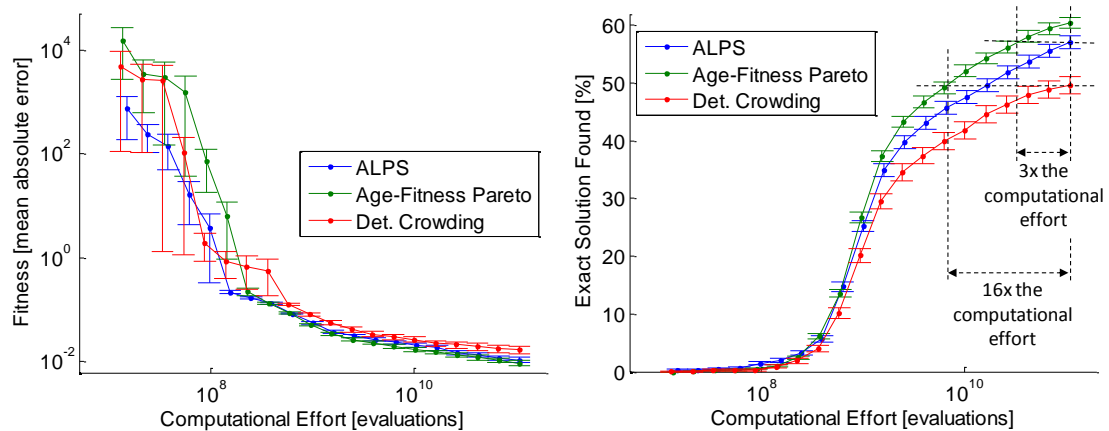
Our first observation is that the fitness trends versus the computational effort of each algorithm are quite similar (Figure 6.3). On average, the ALPS algorithm has the lowest error early on while the Age-Fitness Pareto algorithm has the highest error. This difference, however, does not appear to be significant due to the overlapping standard errors.

Later into the evolution, all algorithms converge to similar fitness trends. This suggests that the algorithms are reaching common local optima. The deterministic crowding method does clearly perform worse here as it is the last to converge on to this trend. Near the end however, the average fitness values are very similar, as most runs for all algorithms do converge to the exact solution.

Figure 6.3 also shows the rate that each algorithm identifies the exact target solution. Here we have clear difference and non-overlapping standard errors for each algorithm.

The ALPS algorithm again has the highest exact solution rate early on in evolution. All algorithms show the standard s-shaped convergence rates where computational effort increases greatly for the hardest of the test problems.

Late in the searches, the algorithms begin to plateau at different rates of finding the exact solution. The Age-Fitness Pareto algorithm performed the best, finding the exact solution approximately 5% more often than the ALPS algorithm.



**Figure 6.3. The fitness and convergence rate to the exact solution of the compared algorithms versus the total computational effort of the evolutionary search. The fitness is plotted (left) is the normalized mean absolute error on the validation data set. Fitness is normalized by the standard deviation of the output values.**

Importantly, Figure 6.3(right) further demonstrates that the hardest problems solved by ALPS were solved by the Age-Fitness Pareto algorithm using a third of the computational effort.

The deterministic crowding algorithm, with the added randomized individual per generation, performed worst of the three algorithms. Here, deterministic crowding identified the exact target solution approximately 5% less often than the ALPS algorithm, and approximately 10% less often than the Age-Fitness Pareto algorithm.

The deterministic crowding algorithm used a randomized individual each generation. However, it still performed significantly worse than the other algorithms. This suggests that the performance improvement is not coming solely from increased diversity through random individuals. Therefore, the genotypic age is playing an important role.

Finally, we looked at the amount of solution bloat experienced by each algorithm over the course of the evolutionary searches in Figure 6.4.

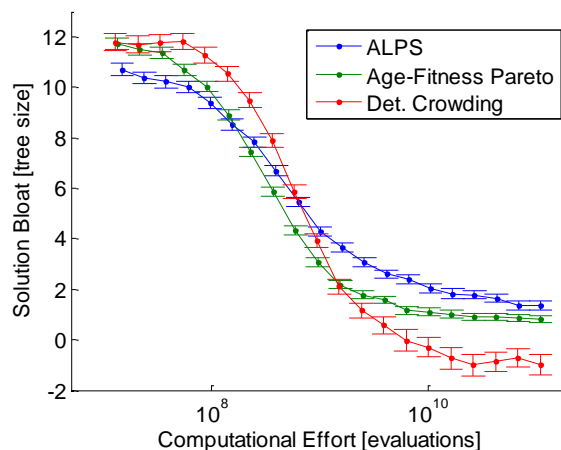
We define bloat as the binary tree size of the best solution in the population minus the binary tree size of the target solution. Therefore, the most bloated solutions have positive bloat values, and overly simple solutions have negative bloat values.

In these results, all algorithms started with high amount of bloated solutions early on in the evolutionary searches. On average, the bloat decreased as the search progressed, and the algorithm converged toward exact solutions.

Interestingly, the deterministic crowding algorithm dropped the most in solution bloat. This suggests that the algorithm is under-fitting – it is stagnating at simple local optima.

In contrast, the ALPS and Age-Fitness Pareto algorithms have similar, more-complex solutions on average, which converge toward slightly bloated solutions. On average, ALPS was the least bloated early on in the evolutionary searches, but bloated the most as the searches progressed.

On average, the deterministic crowding algorithm experience the least bloat,



**Figure 6.4. Solution bloat over the course of the evolutionary search. Solution bloat is defined as the binary tree size of the best individual in the population minus the binary tree size of the target solution. The error bars indicate the**

suggesting that could be under fitting, stagnating at low complexity local optima. The ALPS and Age-Fitness Pareto algorithms instead tended toward slightly bloated solutions on average, which may reflect their higher performance overall.

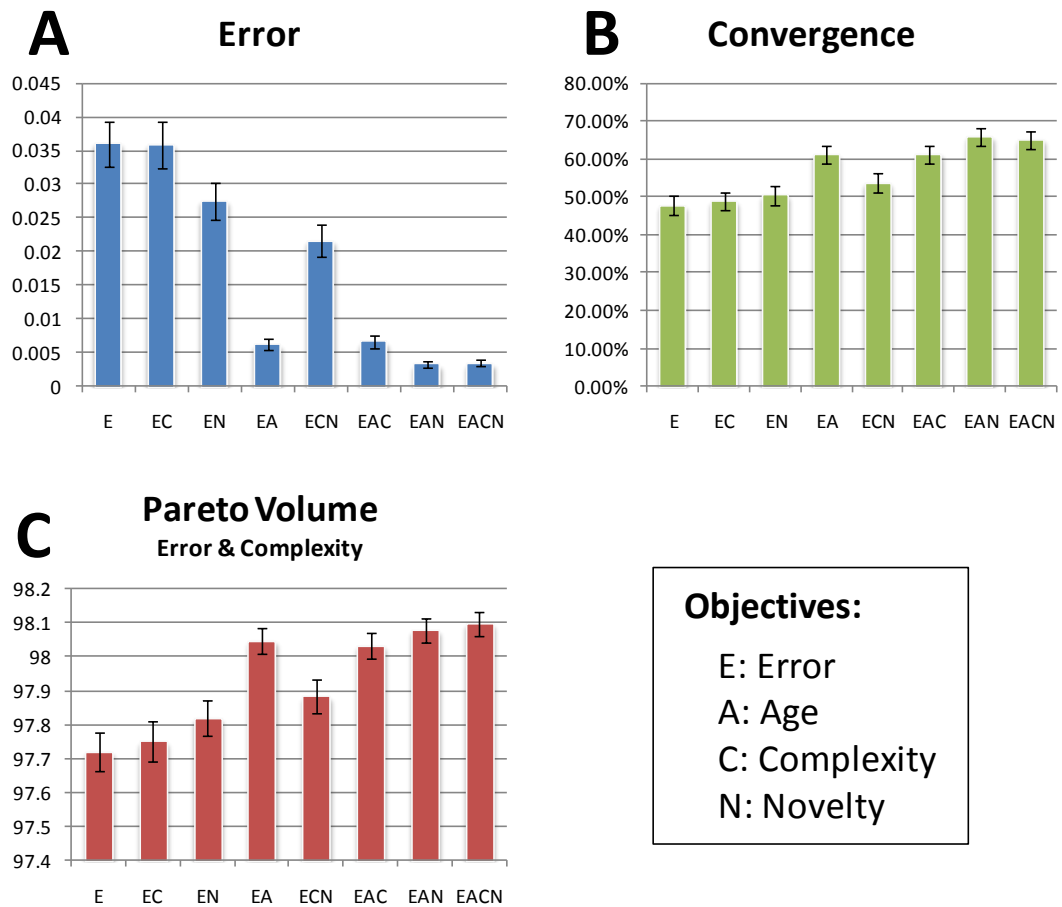
### ***Multi-objective Combinations***

Here we compare the performance of all combinations of secondary objectives: Age, Complexity, and Novelty. The primary objective is Error. This results in  $2^3 = 8$  compared methods. We abbreviate each combination with the letters "E" for Error, "A" for Age, "C" for Complexity, and "N" for Novelty.

For each algorithm we track the best solution over time, and record its final performance. Figure 6.5, summarizes the performance of each on all problems. We consider the error of the best solution (the mean absolute error on a test data set), the convergence (the percent of times that the algorithm identified the exact known solution), and the Pareto volume. The Pareto volume measure the percent of the Pareto space explored by the algorithm. Here, we measure the percent of the Error\*Complexity Pareto space, which are of most interest in the Symbolic Regression problem.

Our first observation from Figure 6.5 is that using the error objective alone ("E") performed the worst for all metrics. This is counter-intuitive; it shows that investing computational effort in any of the three secondary objectives improved performance on error.

Adding complexity to the error objective ("EC") slightly improved convergence and Pareto volume, but otherwise has little impact. Adding novelty objective to error ("EN") we see a substantial improvement in all metrics. Similar to results in the previous experiment, adding age to the error objective ("EA") had the largest impact



**Figure 6.5.** The performance of each combination of the multiple secondary objectives on random symbolic regression problems. Pane (A) shows the mean absolute error on the test data set of the best solution found by each algorithm. Pane (B) shows the convergence rate, the percent of times each algorithm identified the exact solution. Pane (C) shows the percentage of the Pareto space, defined by solution error and solution complexity (the two metrics of interest in the Symbolic Regression), that each algorithm explored.

for a single secondary objective.

Interestingly, combining error, complexity, and novelty ("ECN") improves the performance over novelty ("EN") or complexity ("EC") alone. Combining complexity with age ("EAC") however had no visible change from age alone ("EA").

The two best combinations were error, age, novelty ("EAN") and using all four

objectives ("EACN"). These two methods had equal performance in terms of test set error and convergence. However, adding complexity and using all four yielded slightly higher performance in Pareto volume.

Results in Figure 6.6 show the convergence rate of all combinations versus the problem complexity. The results are split into three panes to better display the difference between the results.

We can see that the performance drops for all algorithms as the problem complexity increases. However, some drop later than others. The differences at the lower convergence rates appear smaller, but the relative difference between the algorithm is actually quite large, with some algorithms achieving 10 times or higher convergence than others.

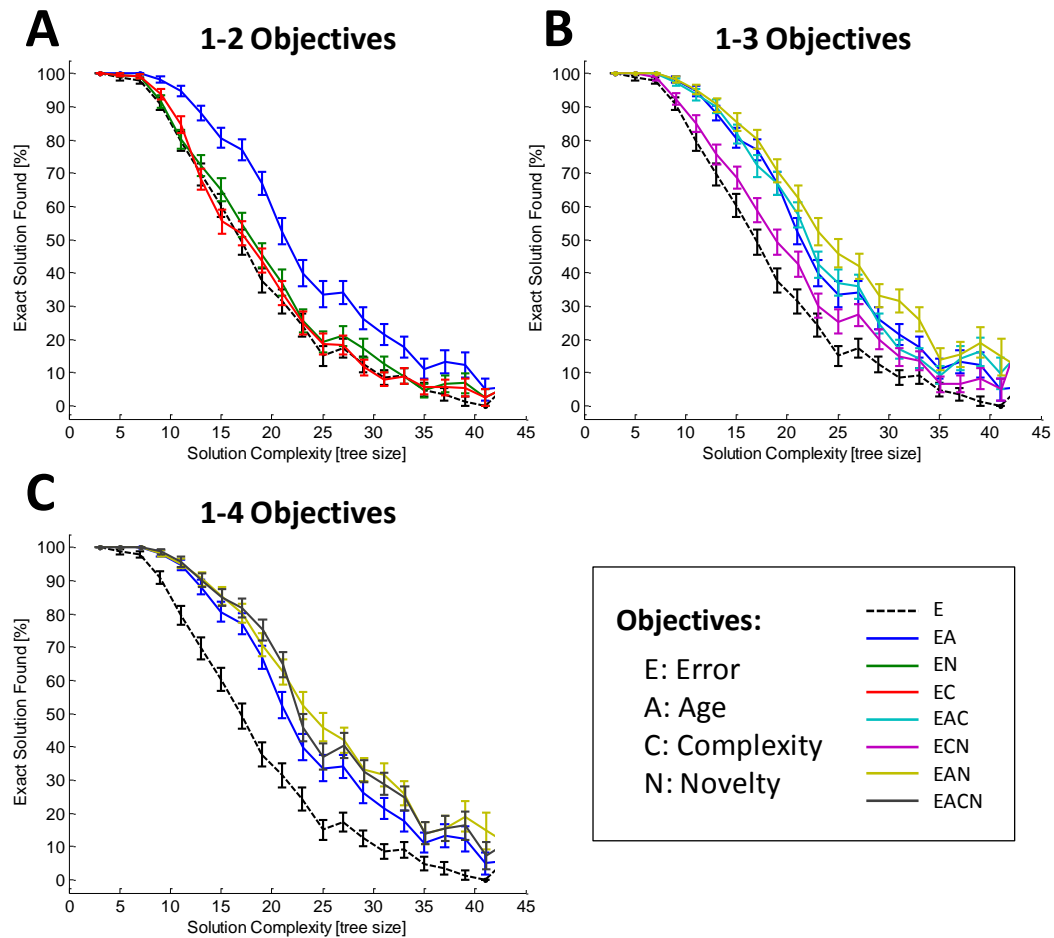
For one and two objectives (Figure 6.6A), the age objective ("EA") stands out showing large improvement over all complexity of problems. Error alone performs worst.

For combinations of three objectives (Figure 6.6B), error, age, complexity ("EAC") roughly matches the performance of error and age ("EA"). The combination of error, age, novelty ("EAN") however makes substantial improvement. The improvement also increases with the problem complexity up to complexity of 33.

Finally, all four objectives (Figure 6.6C), ("EACN") performs well, approximately equal to the error, age, novelty combination ("EAN").

An interesting observation from these results is that age has such a large impact. Combining novelty and complexity improves performance, but combining age and complexity has none. However, combining age and novelty does. This suggests that





**Figure 6.6. The convergence (percent of problems where each method identified the exact solution) versus the problem complexity. These results are split into three panes to make the differences more easily identifiable. Pane (A) shows the results for combinations of two objectives plus the single error objective. Pane (B) shows the results for three objectives plus the best 2 objective method and error objective. Pane (C) shows the best of the previous panes with the 4 objective method.**

age is somehow capturing the benefits of complexity and partially the benefits of novelty on its own.

Complexity only appeared to impact the performance of the Pareto volume. Therefore, it may still be useful as a secondary objective for identifying parsimonious solutions and discouraging bloat.

## **Conclusions**

This chapter looked at using secondary objectives to improve the performance of optimizing a primary objective. Previous research has shown that traits such as genotypic age can be used to greatly improve performance in genetic programming.

We first tested explicitly optimizing for age in a multi-objective search. The Age-Fitness Pareto algorithm selected solutions based on both low error and low genotypic age. Results on randomly generated symbolic regression problems indicate that this approach finds the exact target solution substantially more often than previous methods over a range of target problem complexities and dataset dimensions.

We then looked at two other secondary objectives: complexity and novelty. We tested the performance when combining all combinations of the three secondary objectives. Results showed that the age objective had the largest impact for a single objective. Performance improved slightly more when using novelty and age.

The two best combinations were error, age, novelty and using all four objectives. These two combinations were similar in performance, but adding complexity slightly improved the percentage of the Pareto volume explored.

## CHAPTER 7. PRIOR MODELS AND SEEDING

### **Summary**

We investigated several methods for utilizing expert knowledge in evolutionary search, and compared their impact on performance and scalability into increasingly complex problems. We collected data over one thousand randomly generated problems. We then simulated collecting expert knowledge for each problem by optimizing an approximated version of the exact solution. We then compared six different methods of seeding the approximate model in to the genetic program, such as using the entire approximate model at once or breaking it into pieces. Contrary to common intuition, we found that inserting the complete expert solution into the population is not the best way to utilize that information; using parts of that solution is often more effective. Additionally, we found that each method scaled differently based on the complexity and accuracy of the approximate solution. Inserting randomized pieces of the approximate solution into the population scaled the best into high complexity problems and was the most invariant to the accuracy of the approximate solution. Furthermore, this method produced the least bloated solutions of all methods. In general, methods that used randomized parameter coefficients scaled best with the approximate error, and methods that inserted entire approximate solutions scaled worst with the problem complexity.

### **Introduction**

A common challenge in genetic programming is how to take advantage of prior knowledge and expert knowledge. Utilizing expert knowledge could be used to find solutions that are more interpretable or reliable in their applications (Moore and White 2006; Casey, Bill et al. 2008). Perhaps most importantly however, expert knowledge could be used to scale genetic programs to solve increasingly complex problems

(Banzhaf and Miller 2004) – freeing new evolutionary runs from having to reinvent all past knowledge from scratch over and over.

In this chapter, we explore one of the more general forms of expert knowledge: reusing established or prior solutions to solve a related problem at hand. For example, if we had a model of the metabolic network in a yeast cell, how could we reuse this model to find the metabolic network of a mammalian cell using an evolutionary search? We can generalize this task in genetic programming as the problem of reusing any previous solution that has the same basic problem structure or tree encoding for a new problem.

We define *seeding* as the reuse of a prior knowledge solution by introducing all or any part of its encoding into the population during a new evolutionary run. By injecting genes from a prior knowledge solution, seeding is effectively biasing the evolutionary search toward solutions that use ubiquitous features of the related solution (Mohammad-Reza and Mohammad 1997), even though solutions to the new problems may look very different at a higher level.

There are many potential approaches to seed the solutions in an evolutionary search. Here, we examine six general seeding approaches: injecting prior solutions in their entirety into the population, injecting pieces of the prior solution, injecting entire solutions but with randomly rearranged and shuffled versions of the prior solution, and finally each of these the methods again with either the optimized parameter values from the prior solution or randomized parameters.

We measure the impact of each method on randomly generated problems over one thousand evolutionary runs each. We simulated expert knowledge for these random problems by simplifying and approximating their exact solutions.

In the following sections, we overview back ground information in symbolic regression and seeding, describe each seeding method in greater detail, compare their results in fitness, convergence, and bloat, and end with discussion and conclusions.

## **Background**

### ***Symbolic Regression***

See the description in the section "Symbolic Regression," on page 4.

### ***Equation Complexity***

We define the *complexity* of an equation to be the number of nodes in the equation's binary parse tree. More complex equations are more difficult to find because the evolutionary search must build and optimize a larger solution.

Past results show that the performance of symbolic regression depends heavily on the complexity of the exact target equation (Schmidt and Lipson 2005; Schmidt and Lipson 2006; Schmidt and Lipson 2008). Therefore, we consider the complexity of the problems in our experiments and how the performances of different methods change as target complexity increases.

### ***Convergence***

We define convergence in symbolic regression as when the evolutionary search identifies the exact target solution as the top ranked solution in the population without overfitting.

We test for convergence when generating our final results using a cross validation dataset. The validation dataset has a much wider range of input values than the training dataset used for fitness calculations. This helps distinguish between overfit solutions and exact fits. If the error on the wider cross validation dataset is near zero,

we consider the equation to be converged.

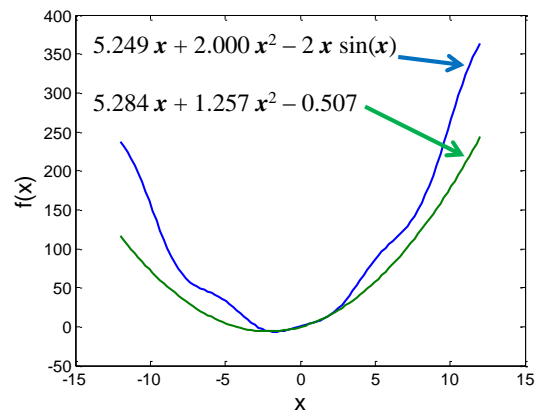
The concept of convergence assumes that there is an exact and general equation underlying the system producing the experimental data. There may be cases however, where no underlying equation exists.

### Seeding Methods

There are many different forms of expert knowledge and ways of incorporating it into an evolutionary search (Moore and White 2006; Casey, Bill et al. 2008). Here, we consider one general form of prior knowledge where we have a prior solution to a simpler problem, or an approximate solution to a more complex problem.

We consider six different policies for using a prior approximate solution: seeding the population with the full solution, seeding with random shuffles of the full solution, a mutation operator for injecting building-blocks of the approximate solution into the population, and finally, using either randomized or optimized parameters for each of these methods (see Figure 7.1).

	Optimized Coefficients	Randomized Coefficients
<b>Whole Equation Seed</b>	$f(x) = 1.3335 x^2 \cos(x - 0.4232)$	$f(x) = \alpha x^2 \cos(x - \beta)$
<b>Shuffled Equation Seed</b>	$f(x) = (x - 0.4232) \cos(1.3335 x^2)$	$f(x) = (x - \alpha) \cos(\beta x^2)$
<b>Building Block Seed</b>	$(x - 0.4232), \cos(1.3335 x^2), 1.0 x^2$	$(x - \alpha), \cos(\alpha x^2), \alpha x^2$



**Figure 7.1. Example seed equations for each method (left) and an example randomly generated target equation plotted next to the automatically generated approximate equation (right).**

### *No seeding*

In the no seeding case, we use an ordinary evolutionary search with a completely random initial population and operators. Variance is introduced solely through random mutation and crossover, and individuals are selected based only on their fitness.

### *Approximate Equation Seed*

In the approximate equation seed, we introduce exact copies of the approximate into the initial population. Only a few equations are seeded to maintain the initial population diversity. In our experiments, we introduce one approximate equation copy for every 10 random initial solutions.

This is the most straight-forward method for using a prior model. The idea is that evolution will use the seeded equations if it likes and will adapt it to the exact model of the system.

There is a potential danger to this method however, in that the approximate solution may trap the evolutionary search in local optima; particularly if the seeded equation is a local optima itself. In the worst case, the evolution fixates on the seeded solution, losing diversity, and is unable to improve upon it.

### *Shuffled Equation*

Instead of seeding with the exact approximate equation, we could instead introduce slightly randomized and rearranged version of the approximation. In the case where the approximate equation is a local optima solution, randomly shuffling its sub-expressions would effectively produce random solutions; but random solutions composed of the same parts of the approximate solution.

Random shuffles of the approximate solution should have roughly the same fitness

distribution as ordinary random solutions, but will still introduce all parts of the approximate equation into the initial population. The idea is that the evolutionary search can recombine these shuffled solutions if beneficial, but will not be immediately placed into a local optima solution.

We implement the shuffling by performing two random shuffles of the approximate solution (or until the fitness changes since shuffles could be neutral). A shuffle consists of picking two random sub-trees of the equation's binary parse tree, and exchanging them.

Though shuffled equations are less likely to push the evolution into local optima, it may not be the best use of the approximate equation. The random shuffles could destroy important parts of the solution, or may be deleterious to the other shuffled components making them difficult to evolve from.

### ***Building block Mutation***

The third method we consider is injecting only individual parts of the approximate equation into the population. We call these parts the building blocks defined by the approximate equation.

We define the set of building blocks for a particular equation to be all sub-trees (sub-expressions) of the equation's binary parse tree.

We define a new type of mutation operator using the set of building blocks defined by the approximate equation. In addition to typical genetic programming mutations, the algorithm can now replace a sub-expression with one of the building blocks at random.

The idea behind this method is that it may be easier to reuse individual pieces of an



expert model rather than adapt the entire equation at once. This operation provides a more granular method for the evolutionary search to pick and choose the useful components of the approximate equation.

One possible danger of this approach is it could produce more bloated solutions, thereby inhibiting finding a general and parsimonious solution.

### ***Parameter Constants***

Finally, for each of the three seeding methods, we can choose to keep the exact coefficient values used in the approximate solution, or randomize these coefficients.

Randomizing the coefficients is one way to deter or delay the possibility of falling into a local optima based on the seed, such as in the whole equation and shuffled equation seed.

The downside of randomizing the constants, however, is that the evolutionary search must always refit them if used. So, randomizing the parameter coefficients does discard some of the prior information contained in the approximate equation.

## **Experiments**

### ***Test Problems***

We used randomly generated problems to evaluate the performance of each seeding method. While random equations do not always resemble real-world applications of genetic programming and symbolic regression, they do provide a base or average case for comparison. Additionally, we can vary and control the complexity of the equations and effectively the difficulty of the evolutionary search.

For symbolic regression, we can produce a random equation in the same way we generate initially random population of equations. We generated one-dimensional equations and then sampled them over the range  $[-2,2]$  to produce synthetic experimental data as would ordinarily be used with symbolic regression. Additionally, we generated a larger test data set over the range  $[-10,10]$ . We use this data set for reporting the performance and convergence rates of each method in our results.

We generated 100 random symbolic equations and corresponding datasets. We then ran each method on the same random problems ten times for each equation.

We also generated the random target equations such that their complexities were evenly distributed. We measure the complexity of an equation as the number of nodes in its binary parse tree. We also perform symbolic simplification of the equation beforehand so that redundant or cancelling terms do not exaggerate the complexity measure.

The random target functions are then evenly distributed between complexities 5 to 35 (or 5 to 35 nodes). Therefore, each seeding method evolves to solve each complexity of target equation approximately 30 times.

### ***Expert Knowledge in Random Problems***

We are using random target equations to generate random problems for testing our seeding methods. Therefore, we need a method for producing expert knowledge for each randomly generated problem. Since we are generating the random problems with a random equation, we know the exact solution to each problem. This allows us to generate approximate models that are equivalent to an expert-derived approximate model, or perhaps an expert derived model of a slightly simpler problem.

We generate the expert knowledge model based on the randomly generated target equation. We first want to approximate this equation so that we aren't giving the exact answer for every problem. To do this we take a randomly generated target equation, and select a random sub-expression that contains at least one operation and is not a leaf node. We then set this sub-expression equal to a random constant.

This creates a simpler and distorted version of the exact target function; however, the output of this function may be drastically different. To be considered an expert knowledge equation, the equation should at least also mimic the general features in the output of the exact target equation.

To mimic an expert derived approximation, we take this simpler equation and refit all of its parameters via nonlinear regression so that it fits the more complex target equation as closely as possible.

The end result is a simpler, but useful approximate model that resembles the target equation that should still have a good initial fitness during evolution. An example is plotted in Figure 7.1. This equation still contains much of the exact structure of the target equation, and is potentially useful for the evolutionary search.

### ***Experimental Setup***

We used the fitness prediction algorithm (Schmidt and Lipson 2005; Schmidt and Lipson 2006; Schmidt and Lipson 2008) to search the space of symbolic equations. Deterministic crowding was used for selection (Mahfoud 1995), with 1% mutation probability and 75% crossover probability. The encoding is an operation list acyclic graph with 64 nodes (Schmidt and Lipson 2007). The operation set contained addition, subtraction, multiply, sine, and cosine operations.

The fitness predictor population contains 1280 predictors, distributed over 80 cores. The fitness predictor subset size is 128 data points. Predictors are also evolved using deterministic crowding, but with 10% mutation and 50% crossover.

## **Results**

We executed 1000 trials per seeding method over 100 randomly generated target equations. We tracked the best solution in each generation, measuring its fitness, convergence, and bloat over the evolutionary run. Fitness and convergence were calculated using a withheld test dataset that spanned a larger input range than the training data set.

### ***Time to Convergence***

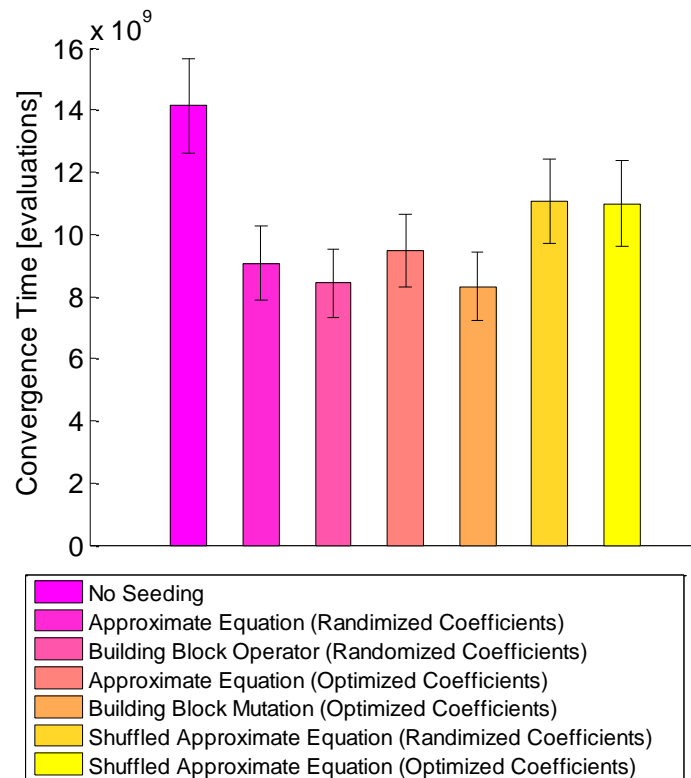
The time to convergence is the total computational effort for each method to find the exact target solution in the evolutionary search. Figure 7.2 compares the convergence time for each seeding method, averaged over all target equations and evolutionary runs.

Time to convergence measures only the runs that did indeed converge. Therefore, it is a measure of the best cases for each method; comparing, potentially, how much the evolution can be sped up with each seeding method. It is important to note however that fast convergence is not always good; but, it is a measure of the evolvability.

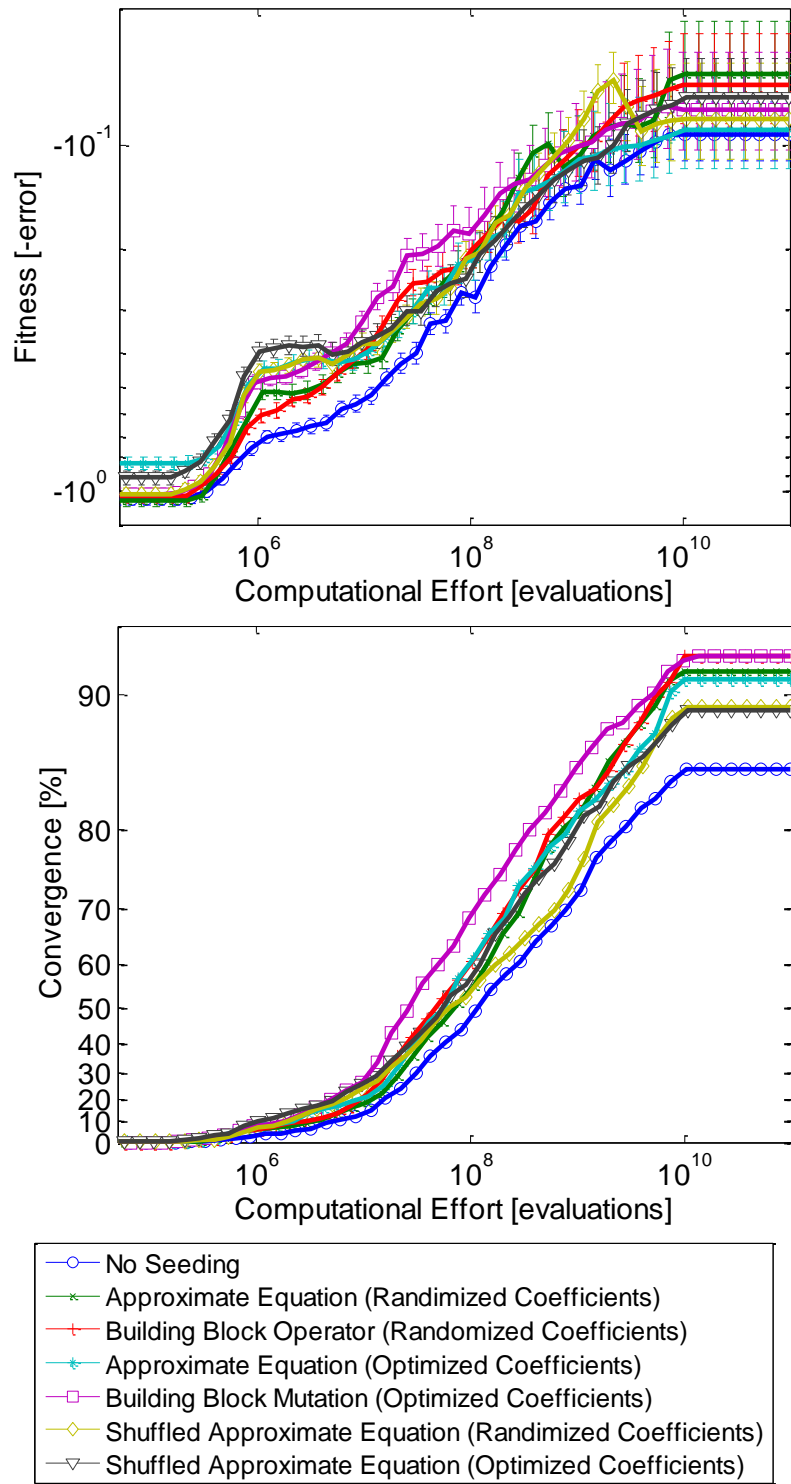
The ordinary evolutionary runs without seeding were the slowest to converge (Figure 7.2) on average. This suggests that all of the seeding methods can speed up the convergence. The next slowest are the shuffled equation seeding methods. This indicates that evolving the randomly shuffled seed equations is the most difficult, but still faster than no seeding at all.

The fastest method to converge is the building block seeding, followed closely by the whole equation seeding. This suggests whole equations and the equation building blocks are easier to evolve than equations from scratch or randomly shuffled equation seeds.

The time to convergence appears to be invariant to using either randomized or optimized parameter constants in the seed. This is particularly interesting because randomized coefficients must always be re-learned or refit. The invariance to the coefficient method indicates that the evolvability and convergence times depend primarily on finding the structure of the equation in the average case.



**Figure 7.2.** The expected time for the evolutionary search to converge to the exact target equation for each seeding method measured in function evaluations (runs that did not converge omitted). Error bars show the standard error.



**Figure 7.3.** The mean fitness (top) and convergence rate (bottom) for each method measured over each evolutionary trial. Error bars show the standard error.

### *Fitness Over Time*

We also tracked the fitness of the top ranked equation over all runs for each method on the withheld cross validation dataset.

We can see that the methods that use whole equations for seeding (the approximate equation and shuffled equation seeding methods with optimized constants) have higher initial fitness as should be expected (Figure 7.3). However, these methods are overcome by the randomized versions later in evolution.

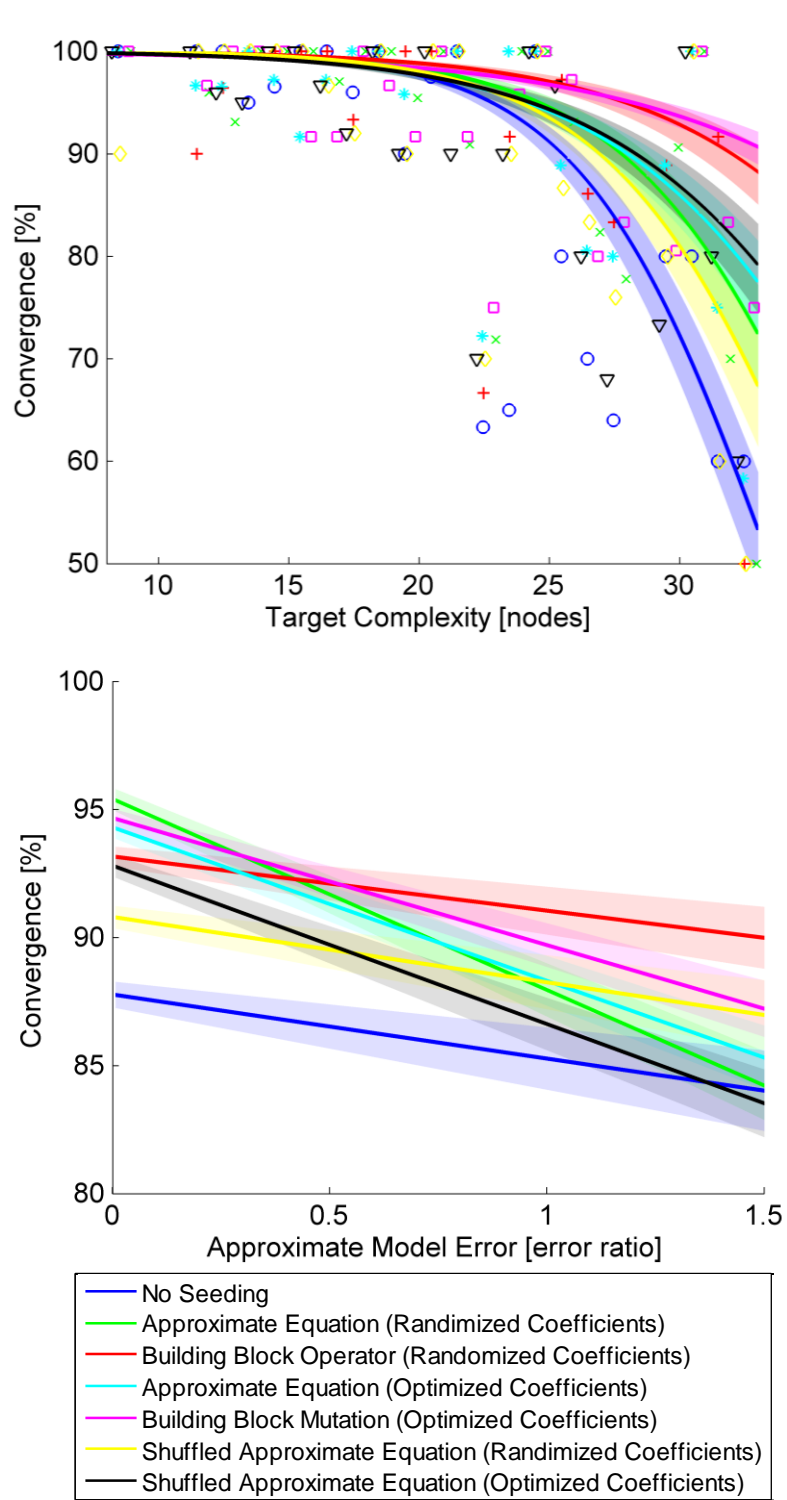
The standard error in fitness increases over time, making it difficult to discriminate between the methods. However, we can pick out some additional general trends. All seeding methods appear to strictly dominate the no seeding method. Also, the most fit solutions at the end tend to be the least fit solutions early on.

### *Convergence Over Time*

The convergence rates over time are more stable than the fitness, making it easier for comparing between each seeding method. The convergence rate shows the percentage of runs that found the exact target solution versus the time (or computational effort) into the evolutionary run (Figure 7.3).

All runs start with zero convergence and increase gradually on a sigmoid trend to their maximum convergence performance. Again, all seeding methods dominate the ordinary non-seeding method. The next worst is the shuffled approximate equations. The highest convergence methods are the building block and equation seed methods.

The building block seeding method with optimized constants stands out the most in Figure 7.3. It converges the soonest, and is tied for the highest convergence rate at the end of each trial with the building block seeding with randomized constants.



**Figure 7.4.** The logistic trends of each seeding method in convergence rate versus target equation complexity (top), and linear trends in convergence versus the error of the approximate seed equation from the target equation (bottom). Error bars show the range based on the standard errors of the trend fit parameters.



### ***Scaling with Complexity***

So far we have only looked at the average performances of each seeding method over all equations. However, the impact of seeding may depend on the different traits of the target functions. Here we break down the performance of each method based on the complexity of the target equation.

Breaking the performances up by the target equation complexity makes the performance trends noisier. Therefore, we use a trend fit to help visualize the differences between each method.

For the convergence versus the target equation complexity, we fit a sigmoid trend curve to each method (Figure 7.4). A sigmoid trend is appropriate for this data since the convergence rate ranges between 0 and 100% depending on the problem difficulty (such as complexity). The sigmoid trend curve has two parameters, the origin slope and the origin offset, making it a low variance trend model.

Based on the sigmoid trends, we want to see which methods drop off in convergence the latest with increasingly complex target equations.

Shown in Figure 7.4, the non-seeding method drops off the fastest. The best performing methods are the building block seeding methods. The remaining methods fall in-between. This result suggests that building block seeding scales the best with the problem complexity; solving the most complex problems more reliably on average.

### ***Scaling with Seed Equation Error***

Next, we look at the convergence rates plotted against the error of the approximate model that is used for seeding. We can view this as the dependence on the confidence or quality of our expert knowledge equation – for example, how does the performance

vary between highly accurate approximate seed equations and inaccurate seed equations.

For this data we fit a linear trend to help visualize the differences between each method (Figure 7.4). This is the most appropriate trend to fit because the seeding equation error does have a dominating influence over the convergence rates. So, we can only pick out the local general trends.

We first notice that there are three methods that appear to be invariant to the approximate model error: no seeding, shuffled equation with random coefficients, and building block with random coefficients (Figure 7.4). This is not surprising for the non-seeding method since it does not use the seed equation. The performance of the other two has the same slope, but higher convergence.

This suggests that the randomized coefficient building block seed and shuffled equation can use parts of the seeding equation even when it is a poor approximation.

It is interesting to note that even the non-seeding method has a slight decreasing trend in convergence with the error of the seed equation, despite not using the seed equation. We generate the seed equation by approximating the exact equation. Therefore, there is a secondary trend in this figure, which is the target equation's sensitivity to approximations. An equation that is difficult to approximate accurately may contain more complex features, thereby making the target equation more difficult to fit in general.

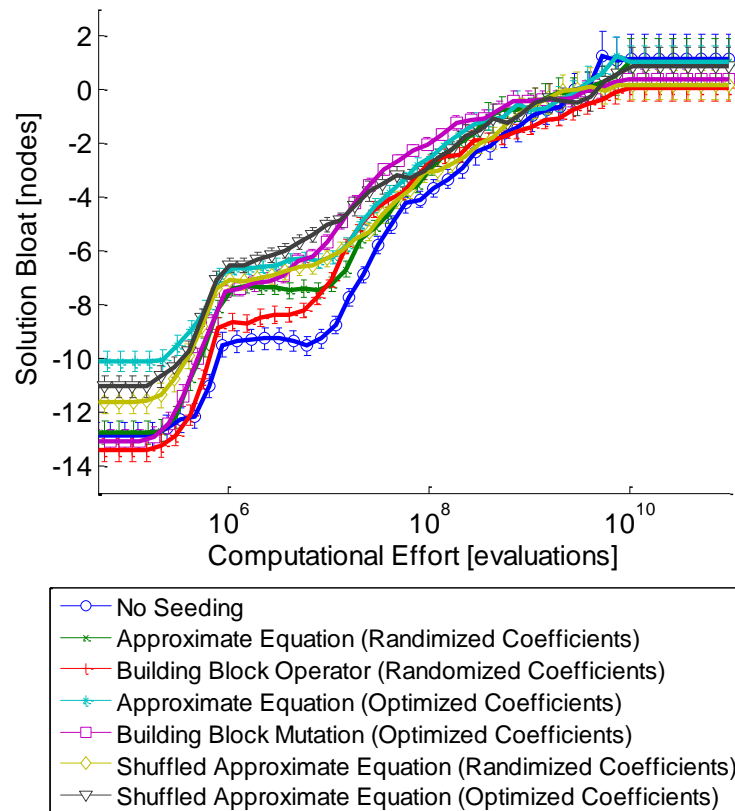
### ***Solution Bloat Over Time***

Finally, we examined the bloat of the top ranked solution of each method in each evolutionary run. We define the bloat as the complexity of the equation (the number of

nodes in the equations binary parse tree) minus the complexity of the target equation. Equations with positive bloat are larger than they need to be while negative bloat means the equation is too small.

We can see that the whole equation seeding methods start off with higher bloat on average (Figure 7.5). This means the seed equations tend to be more complex than the average randomly generated equations. However, all methods converge in complexity toward the target equation complexity over time.

Overall, none of the methods experienced an excessive amount of bloat over time.



**Figure 7.5. The solution bloat of the top ranked solution over the evolutionary runs. Bloat is measured as the top ranked equation’s complexity minus the target equation complexity. Error bars show the standard error.**

However, we can pick out some general trends.

The non-seeding method has the most bloated solutions, and the highest variance in bloat – particularly near the end of the evolution. The building block seeding, in comparison, has the least amount of bloat. This is surprising because the mutation operator with the building blocks provides a means to create additional bloat. Therefore, we suspect that the benefits of the seeding itself dominate this metric, resulting in more exact results on the target solution.

## **Conclusions**

We have explored the effects of incorporating expert knowledge into evolutionary search. We considered a general expert knowledge case, where the expert knowledge consists of an approximate solution or a related solution to the problem at hand. We investigated six seeding methods for utilizing this type of prior expert knowledge: seeding with the whole solution, the randomly shuffled solution, pieces of the solution, and using random or optimized parameter coefficients in each of these three methods.

Our results show that each seeding method can substantially improve the convergence and fitness performance over not seeding. However, different methods scaled differently based on the different traits of the target function.

We found that the building block mutation seeding method converged the fastest among all methods and achieved the highest convergence rates on average for all problems. It also maintained the highest convergence rates for the most complex target equations, and was the most invariant to the error and quality of the seeding equation.

We also found that the seeding methods that used whole equations (no seeding, whole

equation seeding, and shuffled equation seeding) scaled the worst with the target equation complexity. Additionally, the methods that used the optimized parameters (rather than randomized parameters) of the seed equation were the most sensitive to decreasing quality and accuracy of the seeding equation.

While many other possible types of expert knowledge may exist for genetic programming, we conclude that in the case of seeding with a prior solution, it is best to seed with the building blocks of the prior solution, and to randomize the parameter coefficients before seeding.

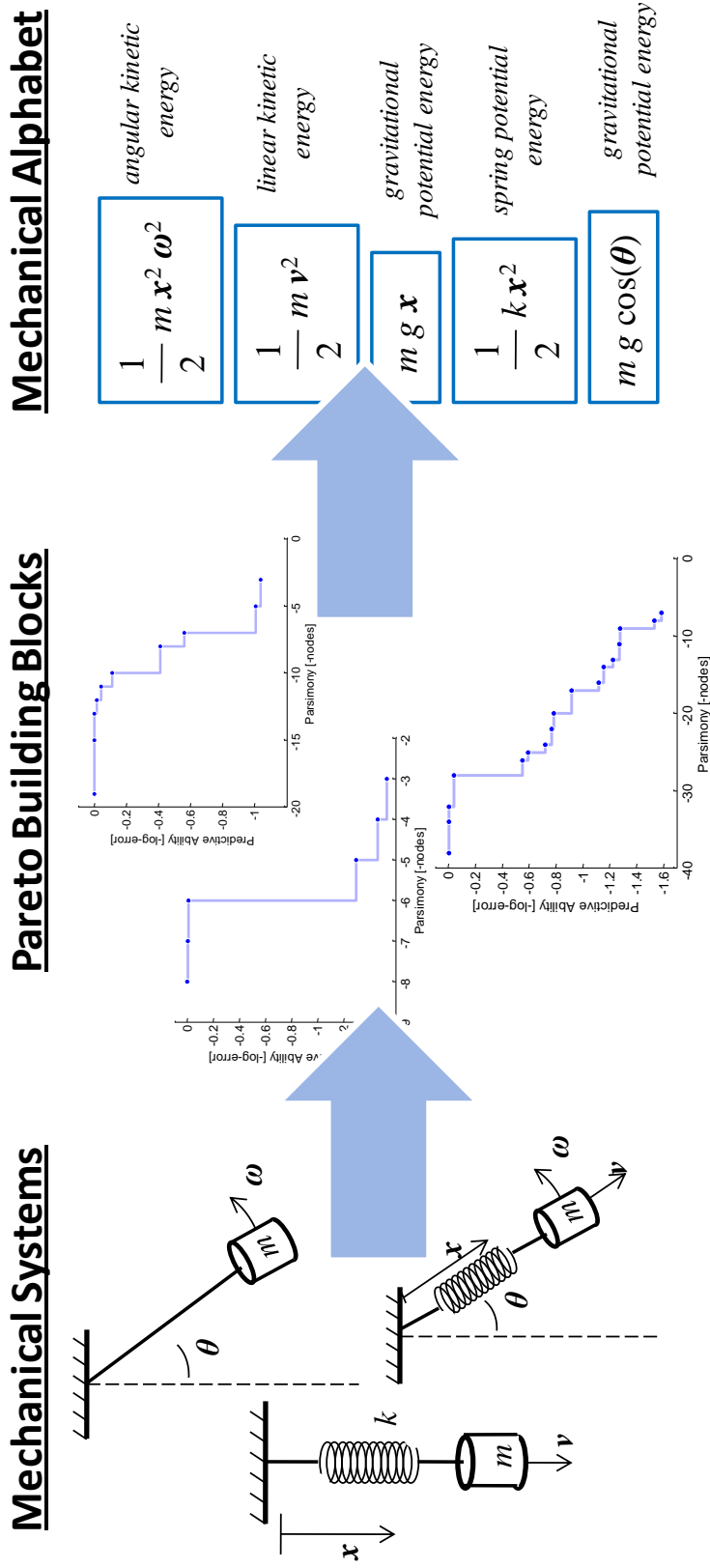
## CHAPTER 8. IDENTIFYING A DOMAIN ALPHABET

### **Summary**

A key to the success of any genetic programming process is the use of a good alphabet of atomic building blocks from which solutions can be evolved efficiently. An alphabet that is too granular may generate an unnecessarily large search space; an inappropriately coarse grained alphabet may bias or prevent finding optimal solutions. Here we introduce a method that automatically identifies a small alphabet for a problem domain. We process solutions on the complexity-optimality Pareto front of a number of sample systems and identify terms that appear significantly more frequently than merited by their size. These terms are then used as basic building blocks to solve new problems in the same problem domain. We demonstrate this process on symbolic regression for a variety of physics problems. The method discovers key terms relating to concepts such as energy and momentum. A significant performance enhancement is demonstrated when these terms are then used as basic building blocks on new physics problems. We suggest that identifying a problem-specific alphabet is key to scaling evolutionary methods to higher complexity systems.

### **Introduction**

Critical to the success of any genetic programming system is the use of a good alphabet of building blocks from which solutions can be evolved efficiently. Typically, GP practitioners will choose generic building blocks based on prior domain knowledge, but this choice may have profound performance implications. An alphabet that is too granular may generate an unnecessarily large search space, while an inappropriately coarse grained alphabet may bias or even prevent finding optimal solutions. Here we investigate a method that identifies an alphabet appropriate for a specific problem domain automatically.



(a) We distill the common mathematical language needed to describe a group of systems using symbolic regression and analysis of their model accuracy/complexity Pareto fronts. We generate experimental data from several related systems such as spring and mass mechanical systems (left). We then use symbolic regression to find several accurate models at varying complexity of equations (middle). Finally, we decompose models on these fronts to individual terms and building blocks. The most frequently used terms and building blocks form an emergent alphabet for describing models of this group of systems (right).

As an example, consider the problem of evolving mathematical expressions that model data collected from an experimental system. If the system is mechanical, its expressions are likely to contain various combinations of trigonometric terms or kinetic energy terms. If the system is biological, then trigonometric terms are unlikely to appear at all; instead, reaction rates and chemical gradient terms such as Hill functions are likely to appear. The availability of appropriate building blocks greatly simplifies both the search space for mathematical models of more complex systems, as well as our conceptual understanding of the results (Holland 2000). A large portion of scientific inquiry has been devoted to unraveling these building blocks by hand. Here, we propose a computational method to explore and learn the language and rules of a problem domain automatically.

Any mathematical equation, or mathematical model, can be decomposed into various combinations of simpler building blocks, such as monomials or trigonometric terms. All of these building blocks are candidates for a common mathematical alphabet of other related systems. Therefore, to build a domain alphabet automatically, we must be able to both generate physically meaningful mathematical models, and be able to identify the nontrivial building blocks from these models.

We use symbolic regression and Pareto analysis to find physically meaningful mathematical models from experimental data. We are interested in finding the most accurate equation at different equation complexities; for example, finding the most accurate model that uses no more than six mathematical operations. These equations are special in the sense that they are both accurate and parsimonious (Kotanchek, Smits et al. 2008) – often consisting of different approximations or elaborations of the physical description of the system.



We break down the models found on the symbolic regression Pareto front into individual terms and building blocks to form a list of candidates for a domain alphabet. The building blocks are extracted by iterating through all sub-trees (sub-expressions) of the equations. Many of these building blocks may not be useful for other systems, such as terms that are overfit to the data or numerical coincidences. Therefore, we need a way to discriminate among the various building blocks.

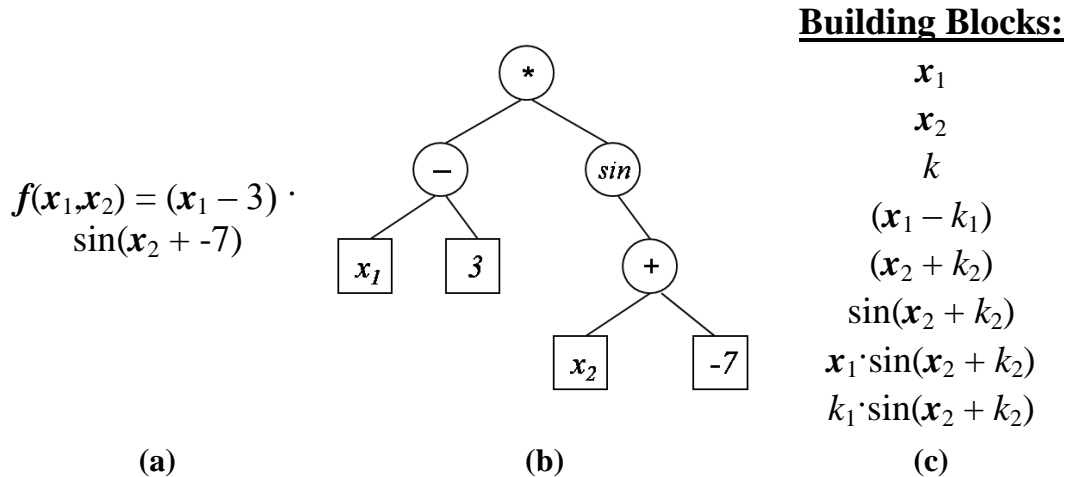
In order to determine which building blocks generalize to other systems in a scientific domain, we need to compare models in two or more systems (Figure 8.1a). We repeat the automated modeling and Pareto analysis to generate candidate building block lists for multiple systems (Figure 8.1b). Finally, we calculate the frequencies that each building block is used in a different system. By considering the frequency and the complexity of a building block itself, we distill the nontrivial building blocks that are the most ubiquitous to return the alphabet of the domain (Figure 8.1c).

## **Background**

### ***Genetic Building Blocks***

*Building blocks* (Holland 1975; Goldberg 1989) are simple expressions which comprise a more complicated solution. While building blocks are most commonly associated with genetic algorithms, they can also refer to sub-trees in genetic programs (O'Reilly 1994; Rosca 1995). For example in symbolic regression, the lowest level building blocks are typically algebraic operations such as add, subtract, multiply, and divide. However, we can also define higher order building blocks such as squaring and multiplying with a constant.

We think of a solution, or equation, as being composed of various types of building blocks (McPhee, Ohs et al. 2008). For example, if we think of an equation as a binary



**Figure 8.2.** Example equation (a), its binary parse tree (b), and all possible building blocks of the equation (c). Building blocks are common sub-expressions or internal components of a system that simplify building a full mathematical model.

parse tree of mathematical operations (Figure 8.2), the set of building blocks for that particular equation contains all sub-trees (sub-expressions) of the tree (O'Reilly 1994).

Knowing the building blocks for a particular problem simplifies human conceptual understanding of the problem (and related problems) by giving higher order meanings and interpretations of the system's mechanics, morphology, or physics. For example, rather than working with cosine operations and a set of variables, a cosine of an angle building block could allow us to work instead with a more meaningful concept, such as the vertical position of a swinging pendulum.

Knowing the basic building blocks of a system ahead of time also greatly simplifies searching for or building a mathematical model to explain its behavior and experimental data – such as done in symbolic regression. Rather than having to re-derive common terms from scratch, over and over again for each model, the algorithm could benefit from the coarser search of assembling higher order building blocks. There are an infinite number of potential building blocks however.

### ***Domain Alphabet***

While there are an infinite number of possible building blocks for any system, we define a *domain alphabet* as the set of building blocks specific to a particular problem, domain, or class of systems that generalize to many similar systems. Domain alphabet building blocks are typically physically meaningful, and are useful for building new models.

Determining the most useful building blocks can be considered to fall under the “credit assignment” problem in machine learning. The credit assignment problem is the task of deciding how to score or weight the importance of individual components of a model when only given entire systems (Grefenstette 1988).

One difficulty to detecting meaningful building blocks is that some building blocks may arise by chance due to overfitting the data, or other numerical coincidences. For example, consider the following equations for two different systems:

$$f = x^2 \cos(x - 1.01) + 2x^3$$

$$g = x^2 \cos(x - 1.02) - \sin(x) + x$$

We would like to be able to identify a term such as  $x^2 \cos(x - 1)$  as a building block given only  $f$ ,  $g$ , and  $x$  values over time (we don’t know the equations in advance) – while rejecting others that are less commonly generated during modeling. The more systems we look at, the less and less likely such a complex building block we be rediscovered repeatedly by chance during evolution. Therefore, finding large repeated building blocks is a strong indication the building block is a nontrivial building block useful throughout the problem domain.

With such information on useful physical terms, the algorithm could reuse them for

analyzing future systems, bootstrapping its knowledge into higher complexity systems. Rather than needing to rediscover common features repeatedly, the algorithm can simplify the problem to the assembly of solutions within the domain alphabet.

### ***Pareto Front***

When generating potential building blocks, we consider the *Pareto front* (Fonseca and Fleming 1993; Fonseca and Fleming 1995) produced by symbolic regression which represents the tradeoff between a model's complexity and its maximum predictive ability for the experimental data. We define parsimony as the inverse of number of terms in the expression and the predictive accuracy as the error on unseen data.

If we consider the relationship between equation complexity and accuracy of fitting the experimental data, there are there two qualitative extremes: extremely complex equations with near perfect accuracy, and simple models with poor accuracy. Equations in-between these two extremes are the most difficult to identify, but their structure tends to be the most meaningful (Kotanchek, Smits et al. 2008).

At certain minimum complexities, the predictive ability tends to increase substantially and then plateau. In other words, there is often a relatively simple model or equation that captures some intrinsic relationships of the system (but perhaps not perfectly). By parsimony arguments, we can reason simpler equations to likely be approximations and more complex equations to be more precise refinements and elaborations of the exact model or overfit solutions to the data.

Though we can't know with certainty what the exact physical model is, it is likely to exist at least partially on this Pareto front. Therefore, when detecting what building blocks may form a general physical alphabet, we consider all building blocks on the Pareto front as candidates for inclusion in the alphabet.

### *Symbolic Regression*

See the description in the section "Symbolic Regression," on page 4.

### **Alphabet Algorithm**

Our goal is to identify the primary mathematical building blocks of a particular problem or domain of systems, thereby building a domain alphabet automatically from experimentally collected data. Our primary challenge is distilling the nontrivial building blocks that generalize to other physical systems for inclusion in the domain alphabet.

Our method has three main steps: (1) finding several mathematical models for two or more related systems, (2) decomposing these models into their constituent building blocks, and (3) identifying the most useful and meaningful building blocks for inclusion in the domain alphabet.

### *Modeling Groups of Systems*

Our first task is to find several system models that define many candidate building blocks. We collect data from several related physical systems (Figure 8.1a) by observing their behavior and dynamics over time. The group of systems should represent qualitatively different dynamics within the same problem domain.

Next, we employ a symbolic regression algorithm (Schmidt and Lipson 2008) to generate several hypothesized mathematical models of each system for varying model complexities.

The output of our symbolic regression algorithm is a small set of equations that lie on the equation accuracy and equation complexity Pareto front for each particular system (Figure 8.1b). The equations on this front are nontrivial in the sense that they represent

the maximum accuracy a model of a given size or complexity can achieve to explain the system's data. The equations on the Pareto front are often different levels of approximation or elaborations of the exact physical system

### ***Extracting Building Blocks***

Now that we have several equations modeling each system, we decompose them into building blocks. For each equation found on each system's Pareto front, we iterate through every sub-tree (or sub-expression) of the equation, adding the sub-tree to our list of potential building blocks (Figure 8.2).

During this process, we abstract away the bulk constants found in each equation and sub-expression to symbolic parameters. For example, we would convert a sub-expression such as  $x + 1.427 \cdot \cos(\theta)$  to  $k_1 \cdot x + k_2 \cdot \cos(\theta)$ . This allows us to later match building blocks between different systems that may only vary by their embedded coefficients.

Additionally, we abstract away variable types based on their units. For example, we consider variables of angles to be equivalent to variables of lengths, but not equivalent to velocities. This allows us later to match building blocks between systems with differences in variable names.

### ***Distilling the Alphabet***

We now have a long list of all building blocks found for each system and must distill this list down to a domain alphabet. We need to identify which are the nontrivial and meaningful building blocks within this list.

If a particular building block exists repeatedly on the Pareto fronts of other systems, it is a strong indication that it is a meaningful building block for the domain alphabet. At

the very least, the building block is certainly useful for forming a parsimonious model in more than one system. This observation forms the basis for identifying the domain alphabet.

If a building block was simply a result of overfitting to the data, it is unlikely to be repeated on the Pareto front of other systems or different datasets because overfit solutions are very sensitive to perturbations and noise in the data. Similarly, if the building block is the result of a numerical coincidence for modeling a particular dataset, it is unlikely that the same coincidence exists in other systems and in their datasets.

Therefore, we can use the frequency that a building block is used on the Pareto fronts as a principle for its generality and importance for a domain of systems. To do this we iterate through all building blocks and count their total occurrences on the Pareto fronts of every other system, and number of times each building block was matched by another.

We form the initial alphabet by rejecting all building blocks that have zero frequency on the Pareto fronts of the other systems.

The second criterion we can use to gauge the importance of a candidate building block is its complexity. Very complex building blocks are much less likely to reoccur by chance or numerical coincidence than simple building blocks. Therefore, we also consider the complexity of the building block when adding it to the domain alphabet.

After calculating the frequencies and complexities of all potential building blocks we examine them graphically to verify the results. We plot each building block as a point on a second type of Pareto space: the building block frequency versus the building

block complexity. As we discovery later, the building blocks on this chart that are both complex and frequently used comprise the physically meaningful domain alphabet.

## **Experiments**

### *The Mechanical Systems*

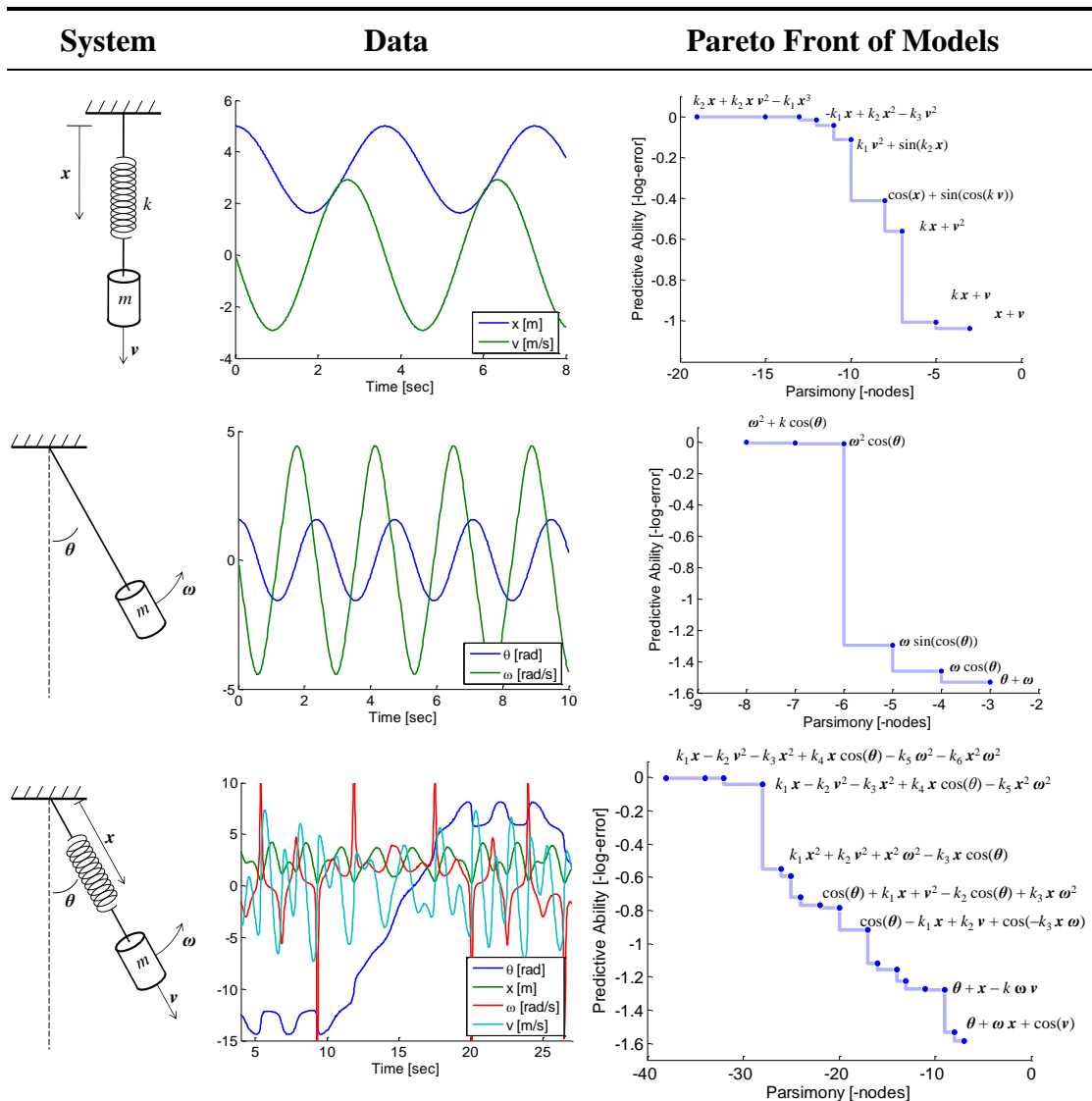
We explore the alphabet building approach using a few simple mechanical systems: a harmonic oscillator, a simple pendulum, and a 2D spring pendulum. These systems are known to have well-defined mathematical models, allowing us to generate data and verify our results. Schematic diagrams of these systems are shown in Figure 8.3.

The harmonic oscillator (Figure 8.3) is a simple conservative system with one degree of freedom. The variables are the mass's vertical position over time and vertical velocity over time. The symbolic regression algorithm identifies several equations modeling the system's kinetic and potential energy over time on the accuracy/complexity Pareto front, including the system's exact Hamiltonian equation.

The simple pendulum (Figure 8.3) is a similar system, but with nonlinear trigonometric terms. The mass's position is measured by the pendulum's angle, and the velocity is the pendulum's angular velocity. Symbolic regression identifies several equations modeling the angular energies over time.

The third system is the more complex 2D spring pendulum (Figure 8.3). Here, the system has two degrees of freedom, two positions, and two velocities measured over time. The dynamics of this systems are more complex, but still tractable with the symbolic regression algorithm.





**Figure 8.3. Summary of the mechanical systems, the collected data of their dynamics, and the resulting models found using symbolic regression on the equation accuracy and complexity Pareto front. Each system was simulated numerically. The symbolic regression algorithm generates a small set of equations for each system. This set is a Pareto front, showing the most accurate equation found for different sizes (complexities) of equations. These equations are used to distill a common mathematical alphabet of building blocks for modeling mass, spring, and pendulum mechanical devices.**

The Pareto front of these systems (shown in Figure 8.3) summarizes the equations that maximize parsimony and accuracy for modeling the experimental data. The terms in the equations are in this sense useful, and may comprise a common physical language.

We simulated these systems numerically by integrating their differential equations. We save the position coordinates and the velocities of each component of the system as the experimental data for use in the symbolic regression algorithm (in Figure 8.3).

### ***Experimental Setup***

Our experiments used the fitness prediction algorithm described in (Schmidt and Lipson 2005; Schmidt and Lipson 2006; Schmidt and Lipson 2008) to search the space of symbolic equations. The selection method was deterministic crowding (Mahfoud 1995), with 1% mutation probability and 75% crossover probability. The encoding is an acyclic graph of 64 operations/nodes (Schmidt and Lipson 2007) and used single-point crossover. The operation set allowed addition, subtraction, multiply, sine, and cosine operations.

We allowed solutions to use up to 64 nodes, each possibly representing five types of mathematical operations, two to four variables, or a parameter constant. Ignoring the possible real values of coefficients, this space contains roughly  $10^{54}$  parameterized genotypes.

We distributed the symbolic regression evolution over 20 quad core computers (80 total cores) (Christian, Marc et al. 2003; Francisco, Giandomenico et al. 2005). The distribution technique partitions the total population of solutions into small local populations residing on each computer (or core). Periodically (every 1,000 generations in our experiments), the total population is randomly shuffled solutions across all computers to better simulate a single large population.

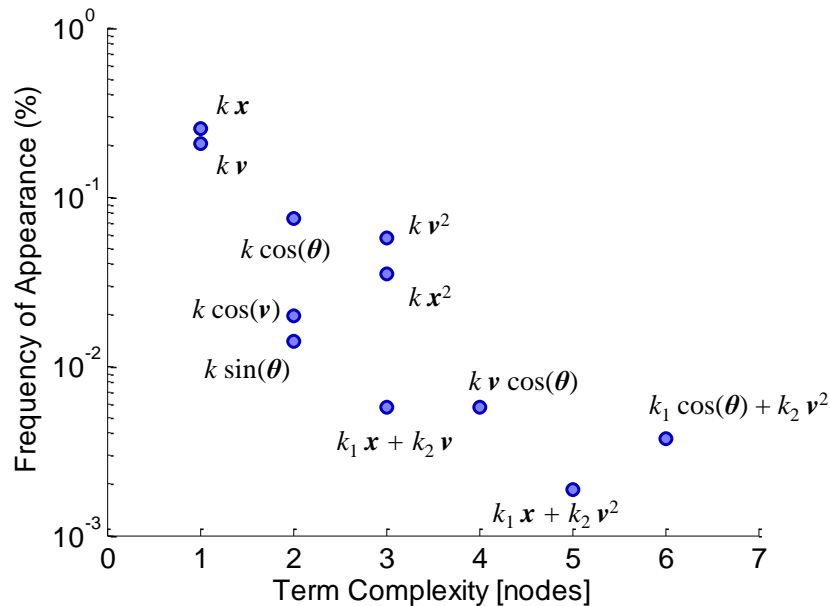
The fitness predictor population contains 1280 predictors, distributed over 80 cores. The fitness predictor subset size is 128 indices to the full training data set. Predictors were evolved using deterministic crowding, with 10% mutation and 50% crossover.

Our fitness calculation rewards equations for modeling the systems kinetic and potential energies as described in (Schmidt and Lipson 2009) as measured over the dataset. The predicted fitness values only calculate over the smaller subset of a fitness predictor rather than the entire data set.

## Results

### *A Mechanical Alphabet*

After building the equation accuracy/complexity Pareto fronts for each system using symbolic regression and decomposing the building blocks for each equation in each system, we plot the frequency of each building block versus its complexity (Figure 8.4).



**Figure 8.4.** The building blocks found for the domain alphabet based on the harmonic oscillator, simple pendulum, and 2D spring pendulum Pareto front models. The most frequent and complex building blocks correspond to the kinetic energy terms for moving masses and potential energy terms for springs and pendula. Building blocks with zero frequency on the Pareto fronts of other systems are omitting and not included in the alphabet.

We can see that single variable terms are the most common building blocks, as well as being the simplest possible building blocks. Not shown in Figure 8.4 are the numerous build blocks that were only found within a single system having zero frequency.

Moving to the next most frequent building blocks, we find  $\cos(\theta)$  and  $k x_2$ . These are pendulum and spring potential energies respectively.

Interestingly, the higher complexity building blocks in Figure 8.4 are the result of matches between inexact equations between the different systems. For example,  $k_1 \cos(\theta) + k_2 v^2$  is an exact building block for the simple pendulum system, but also an approximate solution to the harmonic oscillator.

There are two building blocks in Figure 8.4 which are not exact building blocks for any of the systems, though they are potentially useful approximate building blocks; namely,  $k \cos(v)$  and  $k \sin(\theta)$ . The  $k \cos(v)$  term approximates a kinetic energy and the  $k \sin(\theta)$  term approximates a single variable term. These terms are both low complexity and low frequency however. This hints that these are approximate building blocks and we could elect to reject them after manual inspection.

This result suggests that the terms that are both frequently used and complex tend to be more physically meaningful for inclusion in the domain alphabet, such as trigonometric terms representing potential energies or squared velocities representing kinetic energies.

### ***Utilizing the Alphabet***

One application of the domain alphabet is to simplify the search for forming models of more complex systems. We demonstrate this idea by using an alphabet formed from just the harmonic oscillator and simple pendulum systems to find a model of the more

complex 2D spring pendulum system.

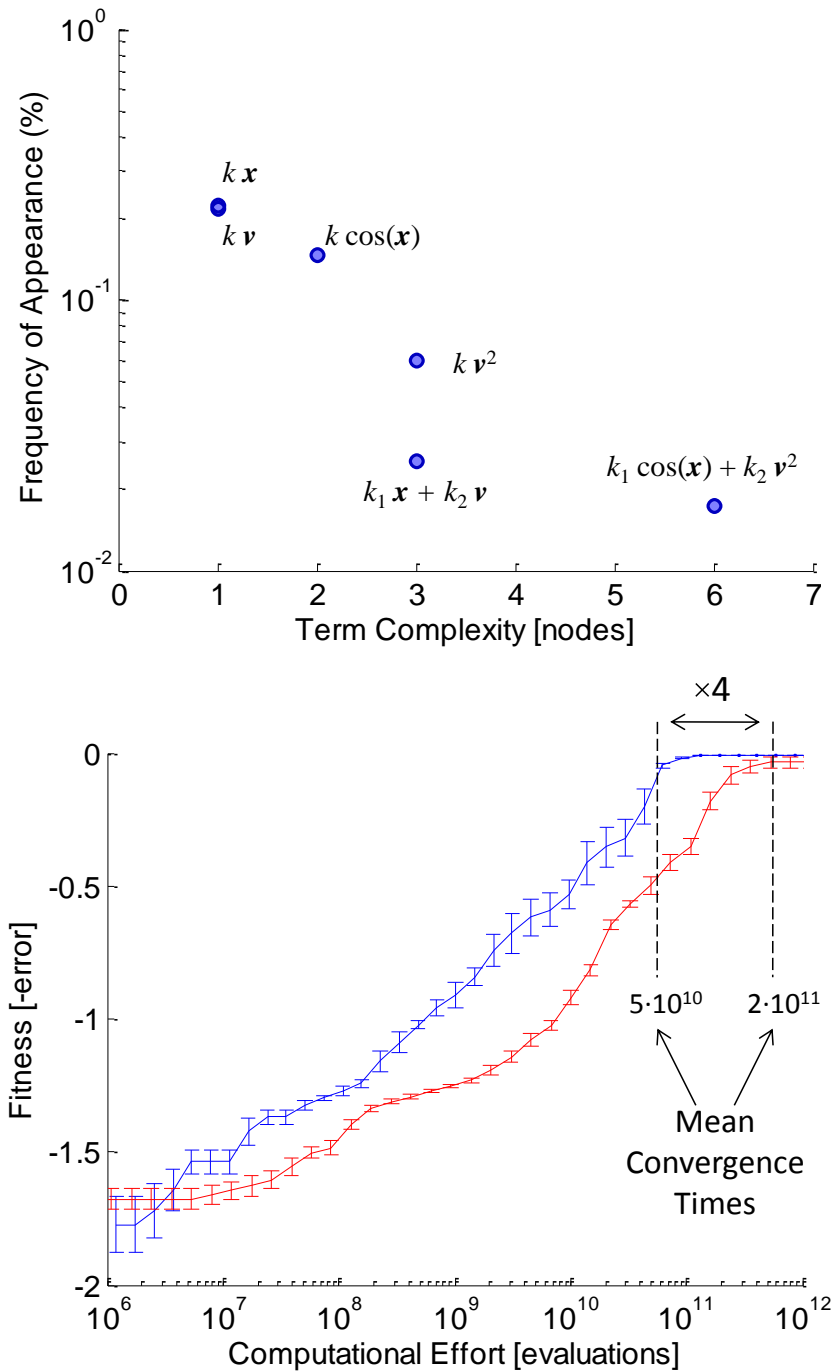
If we repeat the Pareto building block analysis, but now only with the harmonic oscillator and simple pendulum building blocks we obtain the building blocks shown in Figure 8.5a.

There are many ways we could utilize these building blocks in the symbolic regression algorithm. We could introduce them as new functions in the operator set. Alternatively, we could seed the initial population using random combinations of these building blocks.

We chose to introduce a mutation operator that could mutate a sub-expression of an evolving equation to be a random building block from Figure 8.5a. The constant terms in each building block,  $k$ 's, were set to normally distributed random constants at the mutation event. This approach allows the building blocks to be consistently introduced during evolution, but also adapted if necessary. In the case that an alphabet building block is approximate, the evolution can still benefit from using it early on, and adapt its structure later to fit the exact system model.

There are likely much better methods for utilizing the alphabet building blocks in symbolic regression as well as other types of expert knowledge. For the scope of this chapter, we want to show the proof of concept using a simple modification to the program.

Figure 8.5b compares the symbolic regression of the 2D spring pendulum over time with and without the building block alphabet obtained from the harmonic oscillator and simple pendulum. The fitness is shown versus the number of function evaluations, averaged over ten independent trials.



**Figure 8.5.** The impact of using a domain alphabet obtained from simple systems, the harmonic oscillator and simple pendulum, to find the model of a more complex system, the 2D spring pendulum. The alphabet in (top) shows the common building blocks found from the Pareto analysis of only the harmonic oscillator and simple pendulum systems. Allowing symbolic regression to use these terms substantially accelerates the modeling of the more complex 2D spring pendulum system (bottom). Error bars show the first standard error about the mean over ten independent trials.

Using the alphabet substantially improves performance over time, converging sooner onto the exact 2D spring pendulum model. The time to convergence is four times faster using the alphabet building blocks.

This result shows that an alphabet obtained from two simpler systems can be used to accelerate the modeling of a more complex system using symbolic regression.

## **Conclusions**

Identifying a mathematical alphabet is a means to organize and learn the rules and language of a particular scientific field or domain. Alphabets are sets of mathematical building blocks that represent common terms and calculations that pervade different phenomena. Identifying these building blocks helps to generalize our understanding of different systems, and potentially simplify the modeling of future complex systems.

We proposed an automated method to distill the mathematical alphabet directly from experimental data using symbolic regression. The method finds a set of equations for multiple related systems on the accuracy/complexity Pareto front, decomposes these equations into building blocks, and then calculates the frequencies these building blocks occur on the Pareto fronts of the other systems.

Our results suggested that building blocks that are both frequently used and complex tend to be the most physically meaningful to the class of systems; such as spring potentials and kinetic energies. Other building blocks in the resulting alphabet were potentially useful approximations common across multiple systems, such as small angle approximations, but were the least complex and least frequently used.

Finally, we used an alphabet obtained automatically from the harmonic oscillator and simple pendulum systems to accelerate the symbolic regression of the more complex

2D spring pendulum system. The regression using the alphabet found the exact model in one fourth of the computational effort compared to the regression from scratch, suggesting an automated method for scaling into higher and higher complexity systems.



## SECTION II – MODEL REPRESENTATIONS

### CHAPTER 9. DYNAMICAL SYSTEMS

#### **Summary**

This chapter describes a new algorithm for automatically reverse-engineering symbolic analytical models of dynamical systems directly from experimental observations, for the purpose of modeling, control and exploratory analysis. The new algorithm builds on genetic programming techniques used in symbolic regression to infer differential equations from time series data. We introduce the core algorithm for building coherent mathematical models efficiently and then describe its application to system identification. The method is demonstrated on a number of nonlinear mechanical and biological systems.

#### **Introduction**

Many branches of science and engineering represent dynamical systems mathematically as sets of differential equations derived laboriously from basic principles and through experimentation. Until recently, deriving such models has relied on human interpretation or simply fitting data to existing models. In contrast, system identification methods can be used to generate models of a dynamical system automatically from observations. Most system identification methods today are limited to linear systems, or to some classes of nonlinear systems provided the underlying model is known a-priori. Non-parametric methods such as Neural Networks can model nonlinear systems without a preconceived model, but provide little insight into the target system's internal structure. There is a growing need for methods that will be able to generate symbolic models of nonlinear systems without relying heavily on prior knowledge.

Our method uses genetic programming to assemble the exact differential equations that describe an unknown system from scratch (Schmidt and Lipson 2006; Schmidt and Lipson 2006; Schmidt and Lipson 2008). We represent differential equations as an acyclic graph of primitive operations - such as *abs*, *exp*, and *log*, or binary operations such as *add*, *multiply*, and *divide*. The leaves of the graph can represent state-variables of the system or parameter coefficients. We then evolve initially random equations - mutating, recombining, and selecting the best fit equations - until a dominant equation emerges explaining all significant variation in the observed data.

Our algorithm scales favorably into significantly higher-order systems and higher-complexity equations than previous research by coevolving lightweight fitness approximations (Schmidt and Lipson 2008). These approximations adapt to the current population of differential equations in order to predict how well future solutions will explain the data. While these approximations accelerate learning, our results show they also emphasize nonlinear features of the system and mediate solution bloat - biasing the equations to explain basic features before proposing higher-order terms. In ongoing research, we are exploring modeling stochastic systems where manual methods to model and control are most overwhelmed (Schmidt and Lipson 2007).

In the following sections, we provide an overview of our system identification method and describe its adaptation to inferring dynamical systems. We then show new results on a number of classical nonlinear mechanical and biological systems and discuss further applications.

## **Background**

### ***Symbolic Regression***

See the description in the section “Symbolic Regression” on page 4.

**Table 9.1. Fitness prediction algorithm parameters**

Solution Population Size	64 (x 8)
Selection Method	Deterministic Crowding
P(mutation)	0.05
P(crossover)	0.75
Solution Encoding	Operation List (graph)
Max Graph Size	32 nodes
Inputs	7
Operator Set	( +, -, *, /, sin, cos)
Terminal Set	2-dimensional (e.g. x, y)
Crossover	variable position, single point
Fitness Predictor Sample Size	16

### ***Fitness prediction***

Fitness prediction is a new technique to measure how well different mathematical expressions explain experimental data more efficiently and to mediate the pressure to fit multiple aspects of the data (Schmidt and Lipson 2006; Schmidt and Lipson 2008). Fitness predictors only measure fit on a small subset of the data, allowing the algorithm to search solutions faster and build intermediate expressions more easily. However, the data subset is not static: Predictors co-adapt with the solutions to maintain an accurate metric for the fit to the entire data set, so that solutions still move toward a complete model.

See the description in the section “Sub-sample Fitness Predictors” on page 24 for greater detail.

### **Inferring Dynamical Systems**

One form of a mathematical description of a physical or biological system is a set of

ordinary differential equations (ODEs) that describe the time-derivatives of physical positions or chemical concentrations in the system as a function of its current state. Unlike Bayesian networks and information-theoretic approaches, ODEs are deterministic models that describe causal relationships (Bansal, Belcastro et al. 2007) including feedback loops. Terms in the differential equations can correspond to forces such as damping or reactions occurring in the system based on their connectivity. Mathematical models can also be used to predict the behavior of the system in different conditions – such as attracting basins and bifurcations – predictions that are not available in statistical models.

Reverse-engineering ODEs is the task of finding both the correct functional form as well as the parameter constants to fit experimentally collected data. In contrast, many other methods rely on preexisting models to choose a functional form and then use an optimization technique only to fit its parameters (Gardner, di Bernardo et al. 2003; Tegner, Yeung et al. 2003; di Bernardo, Thompson et al. 2005; Bansal, Gatta et al. 2006; Bonneau, Reiss et al. 2006; van Someren, Vaes et al. 2006). However if prior knowledge is limited, it may not be possible to model the system mathematically beyond simple linear models with standard methods (X. Wen 1999). In symbolic regression, both the form and the parameters of the mathematical expression are searched simultaneously in the space of possible algebraic expressions.

Our goal is to algorithmically find an exact mathematical model of some unknown dynamical system. In a system of  $N$  state-variables that we observe experimentally, we must identify  $N$  (possibly nonlinear) differential equations.

### ***Experimental Data***

We can collect data by observing its behavior in time experimentally. We conduct

experiments in silico by integrating a known system model from four initial conditions and observing it for ten seconds. These initial conditions are chosen randomly about its stable nodes or limit cycles.

### ***Handling Noise***

The results shown here were obtained without noise, but in other work we have experimented with noisy data sources. There are various methods for handling noisy time-series data – from filtering and smoothing to spline and polynomial fitting. However, system noise is particularly problematic when calculating numerical derivatives. We use a Loess Fitting (Cleveland and Devlin 1988) both to smooth the data and to calculate time-derivatives of potentially noisy experimental data. We have found empirically this allows yields accurate derivative estimates up to approximately 20% noise (signal to noise). Another approach to handling system noise is to model noise sources directly (Schmidt and Lipson 2007) by incorporating random variables into the mathematical model.

### ***Estimating Numerical Derivatives***

Our approach to finding the differential equations is to measure error directly on the time derivative of each state numerically. There are many methods for numerical differentiation; we have found locally-weighted polynomial fitting (Cleveland and Devlin 1988) to give the most accurate results. At each data point we fit a locally-weighted polynomial, and approximate the derivate numerically as the derivative of the polynomial.

Our fitness function for differential equations then becomes:

$$fitness(s) = \frac{1}{n} \sum_{i=1}^n \left| s(x_i) - \frac{\Delta \tilde{x}_i}{\Delta t_i} \right|$$

where  $s(x_i)$  is the candidate solution (a differential equation) evaluated at  $x_i$  and  $\Delta x/\Delta t$  is the numerically estimated derivative calculated from the data.

There are two key reasons for measuring error on the derivative values rather than their integrals (the measured time-series values). First, the derivative is a lower level comparison and more invariant to small perturbations to the exact solution. For example,  $f'(x) = \sin(x)+0.1$  may be extremely similar to  $f'(x) = \sin(x)$ , but their integrals diverge linearly. Consequently, the fitness landscape is more rugged.

Secondly, and most importantly, measuring error on derivative values rather than integrating allows us to evaluate the fitness of candidate solutions without integrating them. Instead, we can perform point evaluations at arbitrary points within the training data, leading to significantly faster evaluation.

To summarize, we calculate the numerical time-derivative from the data and then use symbolic regression to find a differential equation for each variable individually. We then assemble the final model at the end when we have accurate differential equations for each state-variable.

## **Results and Discussion**

We chose seven two-dimensional dynamical systems that are well studied to demonstrate system identification of various physical and biological models: The *glider*, *bacterial respiration*, *predator prey*, *bar magnet*, *shear flow*, *van der Pol*, and *Lotka-Volterra* models (Strogatz 1994). These systems exhibit many remarkable dynamics (e.g. bi-stability, hysteresis, limit cycles) and are frequently used to understand behavior of other related systems.

**Table 9.2. Inferring various physical and biological dynamical models**

	System	Inferred	Time	Point Evals
<b>Glider</b>	$\dot{v} = -0.05 \cdot v^2 - \sin(\theta)$	$\dot{v} = -0.0499999 \cdot v^2 - \sin(\theta)$	10.219 sec	1.03 B
	$\dot{\theta} = v - \cos(\theta) / v$	$\dot{\theta} = v + (-1 \cdot \cos(\theta)) / v$	5.062 sec	0.50 B
<b>Bacterial Respiration</b>	$\dot{x} = 20 - x - \frac{x \cdot y}{1 + 0.5 \cdot x^2}$	$\dot{x} = 19.994 - 0.998 \cdot x - \frac{1.999 \cdot y}{x + 1.995}$	75.047 sec	7.59 B
	$\dot{y} = 10 - \frac{x \cdot y}{1 + 0.5 \cdot x^2}$	$\dot{y} = 10 - \frac{2.00001 \cdot y}{x + 2.00006 / x}$	30.547 sec	3.09 B
<b>Predator-Prey</b>	$\dot{x} = x \cdot \left( 4 - x - \frac{y}{1 + x} \right)$	$\dot{x} = 4.002 \cdot x - x^2 - \frac{1.003 \cdot x \cdot y}{1.003 + x}$	81.718 sec	8.26 B
	$\dot{y} = y \cdot \left( \frac{x}{1 + x} - 0.075 \cdot y \right)$	$\dot{y} = -0.075 \cdot y^2 + \frac{30.772 \cdot x \cdot y}{30.772 + 30.772 \cdot x}$	290.578 sec	29.38 B
<b>Bar Magnets</b>	$\dot{\theta}_1 = 0.5 \cdot \sin(\theta_1 - \theta_2) - \sin(\theta_1)$	$\dot{\theta}_1 = -\sin(\theta_1) - 0.5 \cdot \sin(\theta_2 - \theta_1)$	11.75 sec	1.19 B
	$\dot{\theta}_2 = 0.5 \cdot \sin(\theta_2 - \theta_1) - \sin(\theta_2)$	$\dot{\theta}_2 = -\sin(\theta_2) - 0.5 \cdot \sin(\theta_1 - \theta_2)$	15.609 sec	1.58 B
<b>Shear Flow</b>	$\dot{\theta} = \cot \phi \cos \theta$	$\dot{\theta} = \frac{\cos \phi}{\sin \phi} \cdot \cos \theta$	3.562 sec	0.36 B
	$\dot{\phi} = (\cos^2 \phi + 0.1 \cdot \sin^2 \phi) \cdot \sin \theta$	$\dot{\phi} = 0.099 \cdot \sin \theta + 0.9 \cdot \sin \theta \cos \phi \cos \theta$	33.859 sec	3.42 B
<b>van der Pol</b>	$\dot{x} = 10 \cdot \left[ y - \left( \frac{1}{3} \cdot x^3 - x \right) \right]$	$\dot{x} = 9.999 \cdot x + 9.999 \cdot y - 3.333 \cdot x^3$	25.547 sec	2.58 B
	$\dot{y} = -\frac{1}{10} \cdot x$	$\dot{y} = -0.1 \cdot x$	0.859 sec	0.09 B
<b>Lotka-Volterra</b>	$\dot{x} = 3 \cdot x - 2 \cdot x \cdot y - x^2$	$\dot{x} = 3 \cdot x - x^2 - 2 \cdot x \cdot y$	4.25 sec	0.43 B
	$\dot{y} = 2 \cdot y - x \cdot y - y^2$	$\dot{y} = 2 \cdot y - y^2 - x \cdot y$	1.063 sec	0.11 B

For each system, we generate time-series data by integrating the known model over ten seconds, from four different initial conditions. We record 100 data points per integration for a total of 400 data measurements. Initial conditions were chosen

randomly near each system's stable nodes or limit cycles.

We distributed the symbolic regression evolution over 4 computers and eight total logical processors using the island model (Francisco, Marco et al. 2003). Every 100 generations, we reshuffle all solutions across all populations. Table 9.1 shows specific settings for the fitness prediction algorithm.

With eight island populations, successful convergence is quite high for these systems. We ran each system once and recorded the time before convergence and the total number of point evaluations (the number of times any function is evaluated in any data point). Results are shown in Table 9.2.

The time to convergence is on the order of one to five minutes over all systems. Most of the differential equations converge in less than 30 seconds. The most difficult equation,  $dy/dt$  in the predator-prey model, took just under approximately 5 minutes. The time to find each differential equation depends primarily on the complexity of its expression and the subtleties of its nonlinearities. For example, in the predator-prey equation, most time is spent finding the  $(1+x)$  denominator.

It is important to note that the algebraic form and parameter values may not exactly match the known model. For example, in the shear flow mode, the algorithm finds a trigonometric transformation of  $\sin^2+a*\cos^2$  to  $a-(1-a)*\cos^2$ , which is equivalent. Additionally, while the known models use precise parameter constants, such as 0.05, the algorithm usually finds close approximations to these constants, such as 0.4999. We could reduce this by running nonlinear regression on the final model to polish off its parameters. Some amount of inaccuracy in the parameters may however be the result of artifacts in the numerical differentiation.



## **Conclusions**

We have proposed a new method for building mathematical models of dynamical systems automatically. The modeling process utilizes symbolic regression using fitness prediction to build differential equations from experimental data.

Symbolic regression with coevolved fitness prediction allows the algorithm to find coherent models reliably in multi-dimensional systems. Fitness predictors specify a small subset of the total training data, effectively focusing regression on a smaller number of features at any given time. In parallel, fitness predictors coevolve to maintain accurate fitness predictions with respect to the cumulative dataset mediate solutions drifting too far away from objective gradient. In this fashion, predictors both reduce computational effort allowing the algorithm to find solutions faster and allowing regression to explore more diverse function-space.

Applying this algorithm to system identification allowed us to infer a number of nonlinear physical and biological systems directly from data.

## CHAPTER 10. IMPLICIT EQUATIONS

### Summary

Traditional Symbolic Regression applications are a form of supervised learning, where a label  $y$  is provided for every  $x$  and an explicit symbolic relationship of the form  $y = f(x)$  is sought. This chapter explores the use of symbolic regression to perform unsupervised learning by searching for implicit relationships of the form  $f(x,y) = 0$ . Implicit relationships are more general and more expressive than explicit equations in that they can also represent closed surfaces, as well as continuous and discontinuous multi-dimensional manifolds. However, searching these types of equations is particularly challenging because an error metric is difficult to define. We studied several direct and indirect techniques, and present a successful method based on implicit derivatives. Our experiments identified implicit relationships found in a variety of datasets, such as equations of circles, elliptic curves, spheres, equations of motion, and energy manifolds.

### Introduction

An implicit equation represents a mathematical relationship where the dependent variable is not given explicitly. For example, an implicit function could be given in the form  $f(x,y) = 0$ , whereas an explicit function would be given in the form  $y = f(x)$ . Implicit equations can be more expressive and are often used to concisely define complex surfaces or functions with multiple outputs. Consider, for example, the equation of a circle: It could be represented implicitly as  $x^2 + y^2 - r^2 = 0$ , explicitly using a multi-output square root function as  $y = \text{sqrt}(r^2 - x^2)$ , or as a parametric equation of the form  $x = \cos(t)$ ,  $y = \sin(t)$ ,  $t = 0..2\pi$ . Our goal is to automatically infer implicit equations to model experimental data.

Regressing implicit relationships can be thought of as a form of unsupervised learning. Ordinarily, Symbolic Regression is used for supervised learning, where a label  $y$  is provided for every input vector  $x$  and a symbolic relationship of the form  $y = f(x)$  is sought. When seeking an implicit relationship of the form  $f(x,y) = 0$ , we are looking for any pattern that uniquely identifies the points in the dataset, and excludes all other points in space.

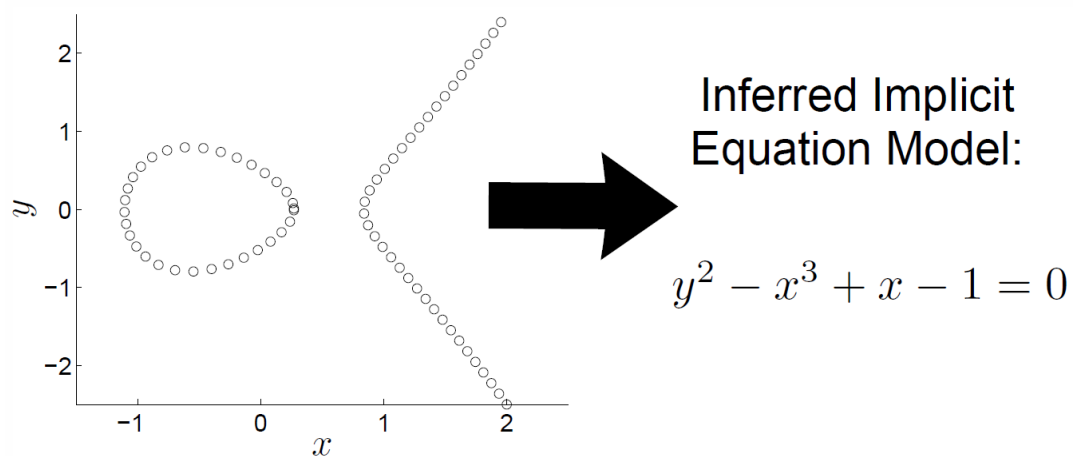
Like clustering methods and other data mining approaches (McConaghy, Palmers et al. 2009), unsupervised learning has the potential to find unexpected relationships in the data (Mackin and Tazaki 2000; De Falco, Tarantino et al. 2002; Hetland and Sætrum 2005). For example, unsupervised learning can create a model from positive examples only, and then use that model to detect outliers that do not belong to the original set. This is important in many practical applications where negative examples are difficult or costly to come by. For example, when training a system to monitor a jet engine, a learning algorithm will typically be trained using sensor data from intact operation only, but will be required to alert an operator if abnormal sensor data is detected.

Implicit equations can also provide deeper insight into the mechanism underlying an observed phenomenon by identifying conservations. For example, when observing a pendulum, an explicit equation can be used to fit the data and thus predict the pendulum's future state based on its current and past states. In contrast, searching for implicit relationships can lead to finding equations of invariants, such as conservation of energy or momentum (Schmidt and Lipson 2009). These conservations can also be used to make predictions, but provide more insight into the underlying principles, beyond prediction.

While symbolic regression has been used to find explicit (Bautu, Bautu et al. 2005; Duffy, Engle-Warnick et al. 2007; Riolo, Soule et al. 2007) and differential equations (Bongard and Lipson 2007), it is not immediately obvious how it could be used to search for implicit equations (Figure 10.1). Symbolic regression ordinarily models and predicts a specific signal or value. In implicit equations, the equation always evaluates to zero over the dataset.

A key challenge is that there are an infinite number of valid implicit equations for any given dataset. For example,  $\sin^2(x) + \cos^2(x) - 1$  is exactly zero for all points in the dataset, but it is also exactly zero for all points not in the dataset. There are also an infinite number of relationships that are arbitrarily close to zero, such as  $1/(1000 + x^2)$ . In order to utilize symbolic regression, we need to devise a fitness function that avoids these trivial solutions.

We experimented with a number of fitness functions for searching invariant equations. We explored minimizing the variance of the function from zero over the dataset while



**Figure 10.1.** Many datasets exist that do not have explicit dependent variables, such as an elliptic curve shown here. Instead, this type of data must be modeled with an implicit equation. We explore using symbolic regression to infer these types of models.

penalizing trivial equations that are zero everywhere, and numerically solving the implicit equation and minimizing its distance to each data point. Due to the difficulty of trivial solutions and susceptibility to local optima, none of these direct methods worked well.

Based on these results, we looked for a different metric that would relate an implicit equation to the dataset. Rather than attempting to model the data points themselves or the zeros of the target function, we decided to look at the gradients of the data. We found that we could derive implicit derivatives of the data variables using an arbitrary implicit equation, and then compare the two. Instead of fitting data points directly, this approach fits line segments (partial derivatives) derived from the data to the line segments (implicit derivatives) of the implicit function.

To test this approach, we experimented on modeling a number of implicit systems – ranging from equations of circles to equations of motion. We found this to be a reliable method for all these systems, whereas the other methods failed to find even the equation of the circle with similar computational effort.

In the remaining sections, we describe the direct methods in more detail, our proposed fitness for arbitrary implicit equations, the experiments and results on modeling implicit systems, and finally, concluding remarks.

### **The Implicit Equation Problem**

The need to search for implicit equations arises when we do not know or do not have an explicit dependent variable in a dataset. Instead, we are given a large vector of data points and our goal is to find an equation that holds true for all of these points. For example, an equation that when solved numerically reproduces the points in the dataset.

An implicit equation has the form:

$$f(x,y,\dots) = 0$$

where  $x$ ,  $y$ , etc. are independent variables of the system. Implicit equations in this form may or may not have an explicit equation in general (it may not be possible to solve for any single variable). However, these equations can be solved numerically or graphically when the equation is known.

Our task is to identify expression  $f(x,y,\dots)$  that satisfies the uniquely for all points in the dataset.

### **Naive Methods**

It might be tempting to search for equations that evaluate to zero for all data points in the dataset. A simple fitness function for this would be second moment or squared-error from zero. The problem with this naive method is quickly obvious however: evolution almost immediately converges to a trivial solution such as  $x - x = 0$  or  $x + 4.56 - y x/y$ , etc. These trivial solutions are zero everywhere and are not particularly interesting or useful for analyzing the data.

We tried a slight modification of this method by adding a test for trivial solutions such as  $0 = 0$ . For each candidate equation, we would perform a quick symbolic simplification to see if the result reduces to zero. Unfortunately, the evolution always converged to more complex identities equal to zero than we could add to our simplification test. For example,  $(x - 1) - (x^2 - 2x + 1)/(x - 1)$  and  $-\sin^2(x) - \cos^2(x) + 1$ , or more complex elaborations of zero identities.

A third method we tried was rewarding the function for being non-zero away from the points in the dataset. In this circumstance, evolution still converged on trivial solutions

that were arbitrarily close to zero over most of the data, but still nonzero away from the data. For example, solutions such as  $1/(1 + x^2)$ , can become arbitrarily close implicit equations over the data, but are still trivial.

Finally, we decided to try numerically solving the candidate implicit equations and comparing with the data points. This method is extremely slow as the numerical solution requires an iterative procedure. It also has serious evolvability problems. Many candidate equations do not have implicit solutions (for example,  $f(x) = 1/x^2$  never crosses zero) which makes finding the numerical solution non-convergent.

We modified this procedure slightly to find the local absolute valued minimum of a candidate equation around each point in the data set, summing the distance from the data points to their minima on the implicit function and the distance of the minima from zero. In the case that there is no local minimum for a data point, we capped the iterated procedure to a maximum distance.

This approach was able to identify implicit versions of simple lines, such as  $x + y = 0$ , and once finding the correct implicit equations in the unit circle dataset (though these solutions were not repeatable). Unfortunately, all runs on more complex dataset, and most runs on the unit circle dataset, became trapped in local optima solutions. A common type of local optima evolved zeros around a part of the dataset (for example  $1/(x + a) - b - y$  can model the left and bottom sides of a circle accurately), but rarely jumped to fit remaining data points.

While this final direct method may be a workable approach with more sophistication, it is far from elegant or efficient. Below, we describe a more direct and greatly more reliable and efficient fitness calculation for implicit equations.

**Table 10.1. A summary of direct methods and their difficulties**

<b>Method</b>	<b>Difficulty</b>
Equations that equal zero at all data points	Trivial solutions such as $0 = 0$ , $x - x = 0$ , etc.
Equations that equal zero near data, but grow with distance	Places too many constraints on the resulting equations
Equations that equal zero but have non-zero derivative	Places too many constraints on the resulting equations
Equations that equal zero but not symbolically zero when simplified	Trivial solutions, just more complex zero identities such as $\cos^2(x^3) + \sin^2(x^3) - 1$
Equations that Equal zero, but nonzero at random point away from data	Trivial solutions such as $f(x) = 1/(100 + x)^2$ , which is non-zero near $x = -100$
Numerically solve equation, measure distance from data points to closest zero	Difficult to evolve, many degenerate equations do not have solutions, and computationally expensive

### **The Implicit Derivatives Method**

The difficulties of the direct methods (Table 10.1) suggest that comparing the zeros of the candidate implicit equation directly is insufficient to reliably find accurate and nontrivial models.



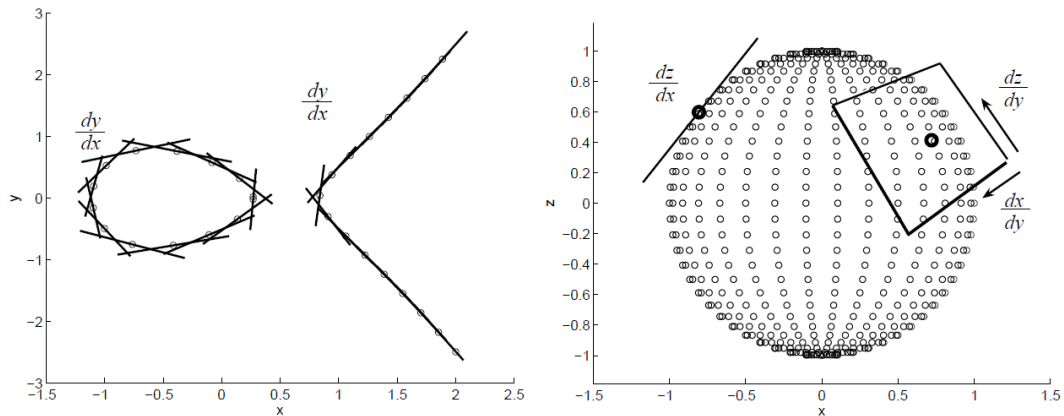
Rather than looking at the individual points, we decided to look at the local derivatives of these points. If the candidate implicit equation is modeling the points in a meaningful way, it should be able to predict relationships between derivatives of each variable. Importantly, we must also be able to measure such a relationship readily from the dataset.

For our method, we propose using the ratio of partial derivatives between pairs of variables (implicit derivatives). The idea is that dividing two partial derivatives of a candidate implicit equation  $f(\dots) = 0$  cancels out the implicit  $f(\dots)$  signal, leaving only the implied derivative between two variables of the system.

For example, in a two-dimensional dataset we could measure variables  $x(t)$  and  $y(t)$  over time. The system's implicit derivatives estimated from time-series data would be  $dx/dy \approx x'/y'$  and  $dy/dx \approx y'/x'$ , where  $x'$  and  $y'$  represent the time-derivatives of  $x$  and  $y$ . Similarly, given a candidate implicit equation  $f(x,y)$ , we can derive the same values through differentiation:  $dx/dy = (df/dy)/(df/dx)$  and  $dy/dx = (df/dx)/(df/dy)$ . We can now compare  $dx/dy$  values from the experimental data with  $dx/dy$  values from a candidate implicit equation  $f(x,y)$  to measure how well it predicts indirect relationships between variables of the system.

Finally, we can use this process in a fitness function for implicit equations. We simply measure the error on all implicit derivatives that we can derive from each candidate equation. In our experiments, we return the mean logarithmic error of these derivatives:

$$-\frac{1}{N} \sum_{i=1}^N \log \left( 1 + \left| \frac{dx_i}{dy_i} - \frac{df}{dy} / \frac{df}{dx} \right| \right)$$



**Figure 10.2. Implicit derivatives can be estimated from unordered, or shuffled data, non-parametrically by fitting a hyperplane or higher-order surface to neighboring points. After fitting the neighboring points, simply take any of the implicit derivatives of the locally fit surface.**

where  $N$  is the number of data points,  $dx/dy$  is a implicit derivative estimated from the data, and  $(df/dy)/(df/dx)$  is the implicit derivative derived from the candidate implicit equation.

### Handling Unordered Datasets

The implicit method can also be applied to unordered and non-time series data as there are several ways to estimate implicit derivatives from experimental data. An implicit derivative is simply a local relation of how two variables covary. In 2D, the implicit derivative is the slope of the tangent line. In 3D, the implicit derivatives lie on the tangent plane. In higher dimensions, they lie on the  $n$ -dimensional tangent hyperplane.

To generalize this procedure for arbitrary unordered data, one can fit a hyperplane, or higher-order surface such as a conic section (Shpitalni, M et al. 1997), to local clouds of data points. From each hyperplane, one can then sample implicit derivatives by taking the implicit derivative of the hyperplane equation (Figure 10.2).

We verified that this procedure works in our experimental datasets by randomly

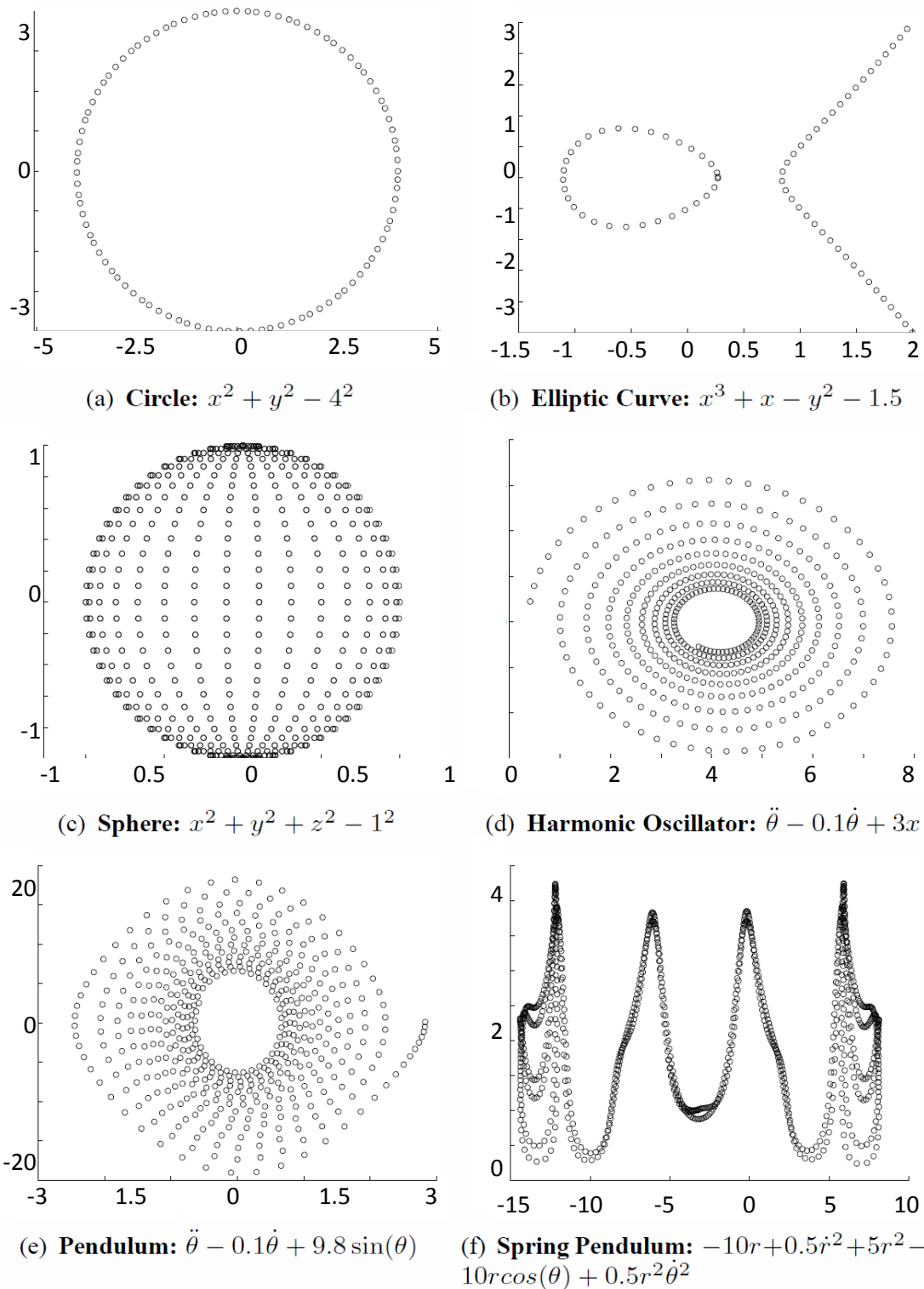
shuffling them and discarding their time ordering. The method regresses the same implicit equations as in our results below using this procedure.

## **Experiments**

We experimented on six implicit equation problems of varying complexity and difficulty (Figure 10.3). The simplest are the equation of a circle and an elliptic curve. These are well-known two dimensional systems with only two implicit derivative ( $dx/dy$  and  $dy/dx$ ) that require implicit equations. A similar but slightly more difficult problem is the 3-dimensional sphere. In each of these systems we can collect data uniformly on their implicit surfaces.

The next three systems are dynamical systems of varying complexity: a simple linear harmonic oscillator, a nonlinear pendulum, and a chaotic spring-pendulum. We simulated single trajectories of each system, recording the positions, velocities, and accelerations for the implicit datasets. In these systems, we are seeking the implicit equation of motion. In the spring-pendulum we are seeking a similar implicit equation, the Hamiltonian, which only uses position and velocity data. The data used for each system is shown in Figure 10.3.

From this data, we estimate the partial derivatives from the data ( $dx/dy$ ) by taking the ratio of the time derivatives. For the circle, elliptic curve, and sphere, we picked an arbitrary time trajectory around their surfaces (two in the case of the elliptic curve). This works because the time component cancels out in the ratio. We could also have fit a local plane to each point to estimate the partial derivatives non-parametrically of unordered data as discussed earlier.



**Figure 10.3. Data sampled from six target implicit equation systems. Data is collected uniformly for the geometric systems. In the dynamical systems, the data is a single simulated trajectory from a random initial condition.**

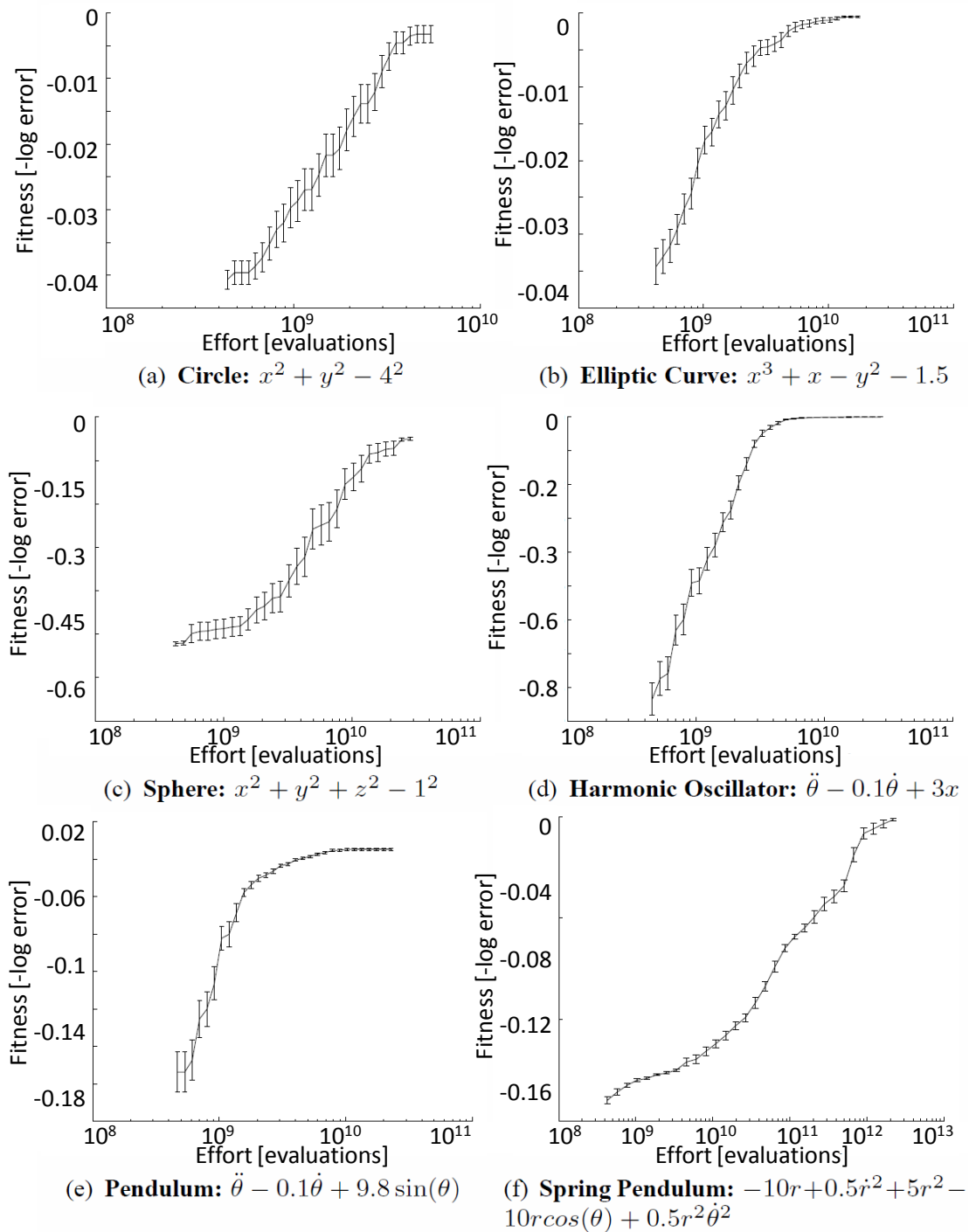
We used a basic symbolic regression algorithm (Schmidt and Lipson 2006) to search the space of implicit equations. We use the deterministic crowding selection method (Mahfoud 1995), with 1% mutation probability and 75% crossover probability. The encoding is an acyclic graph (Schmidt and Lipson 2007) with a maximum of 128 operations/nodes. The operation set contains addition, subtraction, multiply, sine, and cosine operations.

## **Results**

We conducted 20 independent trials on each system, recording fitness values and solutions overtime. Evolution was stopped after a solution converged onto a near perfect solution. Figure 10.4 shows the mean fitness of the top-ranked solution during the evolutionary runs on a validation dataset.

Each evolutionary run identified the correct implicit equation for these systems, although different systems required more computation than others. The circle took less than a minute to converge on average; the elliptic curve, sphere, and pendulum took five to ten minutes on average; and the spring pendulum took approximately one to two hours.

In comparison, none of the direct methods could find solutions to any of these systems, even with considerably more computational effort. In the case of the circle, the implicit derivatives methods obtained the correct solution 20 out of 20 trials in under one minute per trial. In contrast, the direct methods did not obtain the correct solution even once in 20, one hour trials. The best solution found by the direct method over these runs was  $a/(x^2 + b) - y - c = 0$ . In the remaining target systems, the direct methods performed even worse.



**Figure 10.4.** Fitness of the symbolic regression algorithm using the implicit derivatives fitness for each of the six systems. Results are the top ranked solution versus time, averaged over 20 independent trials. Error bars indicate the first standard error.

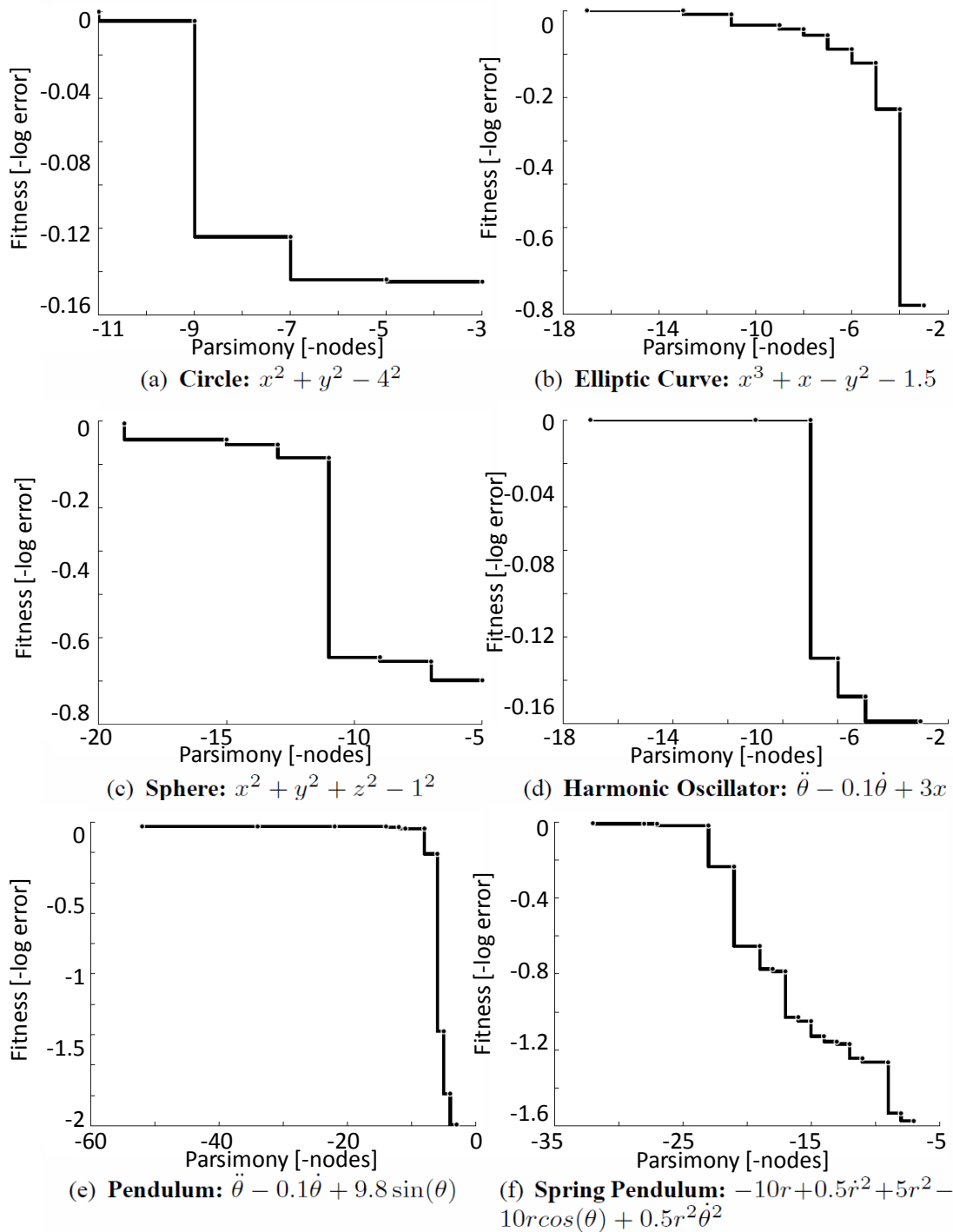
## Implicit Pareto Fronts

Over our experiments, we also tracked the *Pareto frontier* of the implicit equation fitness and complexity for each system (Figure 10.5). This front shows the tradeoff between equation complexity and its ability to model the implicit data (Smits and Kotanchek 2004). Here, we measure the complexity of an equation as the number of nodes in its binary parse tree.

The Pareto fronts tend to contain cliff features where fitness jumps rapidly at some minimum complexity. In the cases where even more complex equations are found on the front, even several times more complex, the improvement in fitness is only marginal.

For each system, the simplest implicit equation to reach the highest qualitative fitness on the Pareto front was the exact target equation. Looking more closely at the higher complexity solutions, we found they were elaborations on the exact solution -- for example, extraneous terms with very small coefficients, perhaps compensating for small errors in estimating the partial derivatives from the data.

We also noticed that simpler and lower fitness solutions on the fronts contained approximations to the exact solutions -- for example, small angle approximations in the pendulum and spring pendulum systems.



**Figure 10.5. The fitness and equation complexity Pareto fronts found for each of the six systems. The exact solutions are the simplest equations to reach near perfect fitness. More complex solutions show elaborations on the exact solution, improving fitness only marginally.**



## **Conclusions**

The ability to search for implicit equations enables searching for multi-dimensional surfaces, equations of motion, and other invariant models in experimental data. However, identifying meaningful and nontrivial implicit equations poses difficult challenges.

We explored several naive fitness methods for rewarding implicit equations to model data. These methods, which considered the individual data points and the zeros of the implicit equations directly, were unable to solve the simplest implicit equations reliably or consistently.

We showed that looking instead at ratios of partial derivatives of local data points provided a reliable search gradient for a variety of implicit systems. This method identified geometric equations such as elliptic curves and 3-dimensional spheres, as well as equations of motions in nonlinear dynamical systems.

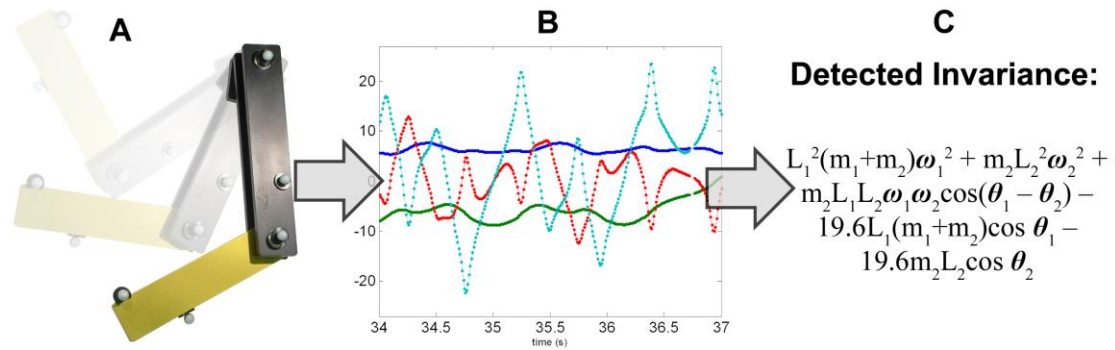
## CHAPTER 11. NATURAL LAWS

### **Summary**

For centuries, scientists have attempted to identify and document analytical laws that underlie physical phenomena in nature. Despite the prevalence of computing power, finding natural laws and their corresponding equations has resisted automation. A key challenge to finding analytic relationships automatically is defining algorithmically what makes a correlation in observed data important and insightful. We propose a principle for the identification of non-triviality. We demonstrate this approach by automatically searching motion-tracking data captured from various physical systems, ranging from simple harmonic oscillators to chaotic double-pendula. Without any prior knowledge about physics, kinematics or geometry, the algorithm discovered Hamiltonians, Lagrangians, and other laws of geometric and momentum conservation. The discovery rate accelerated as laws found for simpler systems were used to bootstrap explanations for more complex systems, gradually uncovering the "alphabet" used to describe those systems.

### **Motivation**

Mathematical symmetries and invariants are known to underlie nearly all physical laws in nature (Anderson 1972), suggesting that the search for many natural laws is inseparably a search for conserved quantities and invariant equations (Noether 1918; Hanc, Tuleja et al. 2004). Automated techniques for generating, collecting and storing data from scientific measurements have become increasingly precise and powerful, but automated processes for distilling this data into knowledge in the form of analytical natural laws have not kept pace. This trend is incommensurate with the rapidly increasing influx of scientific measurements (Clery and Voss 2005; Szalay and Gray 2006) coupled with the growing complexity of systems being studied (Strogatz 2001;



**Figure 11.1. Mining physical systems:** We captured the angles and angular velocities of a chaotic double-pendulum (A) over time, using motion tracking (B), then automatically searched for equations that describe a single natural law relating these variables. Without any prior knowledge about physics or geometry, the algorithm found the conservation law (C), which turns out to be the double-pendulum’s Hamiltonian. Actual pendulum, data and result shown.

Marquet 2002). There is thus a pressing practical need for new forms of scientific data mining (Ra, l et al. 1999; King, Whelan et al. 2004).

The most prohibiting obstacle to overcome in order to search for conservation laws computationally is finding *meaningful and nontrivial* invariants. Here we introduce a new principle for identifying useful analytical relationships. We then demonstrate how a search algorithm based on this principle identifies meaningful analytical relationships in data captured from a variety of physical systems (Figure 11.1).

Our goal is to find natural relationships *where they exist*, with minimal restrictions on their analytical form (i.e. *freeform*). Many methods exist for modeling scientific data: Some employ fixed-form parametric models derived from expert knowledge, others use numerical models (such as neural networks) aimed at prediction. Alternatively, we seek the principal freeform analytical expression that explains symbolically precise conservation relationships, thus helping distill the dataset from correlations into scientific knowledge.

## Method Overview

The established method for search a space of mathematical expressions to minimize various error metrics is known as *Symbolic regression* (Koza 1992), a method based on evolutionary computation (Forrest 1993). See the description of the section “Symbolic Regression” on page 4 for more information.

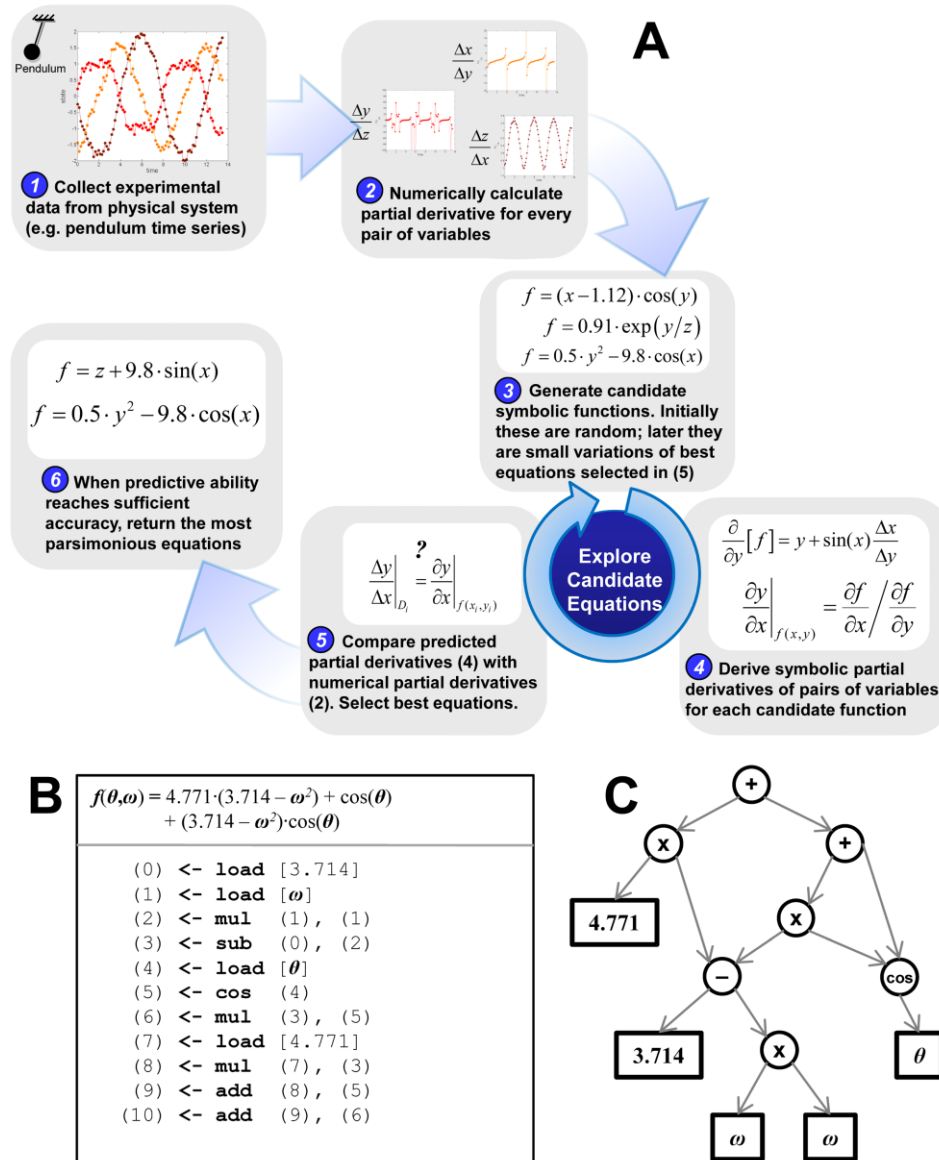
While symbolic regression is typically used to find explicit (Duffy and Engle-Warnick 2002; Elena, Andrei et al. 2005; Cyril and Alberto 2007) and differential equations (Bongard and Lipson 2007), symbolic regression cannot readily find conservation laws or invariant equations. We simply do not know *a priori* what exactly the equations should model or what they should evaluate to, and so a direct error metric is elusive. Rather than trying to model a specific signal, we are trying to detect any underlying physical law that the system is obeying, which may or may not be constant (e.g. a Lagrangian).

A particular challenge is requiring the detected law to be a function of the system’s state while avoiding trivial or meaningless relationships. For any system over the state space  $\underline{x}$ , there are, in fact, infinitely many *trivial equations* over  $\underline{x}$  that satisfy a conserved quantity, such as  $\sin^2(x_1)+\cos^2(x_1)$  or  $x_1+4.56-x_2x_1/x_2$ . Additionally, there are infinitely many arbitrarily-close trivial conservations, such as  $4.56+1/(100+x_1^2)$ . Clearly, we need a more robust principle for distinguishing good conservation law candidates from poor ones, than simply invariance alone.

The identification of nontrivial relationships is known to be a major challenge even for human scientists: Many published invariant quantities have turned out to be coincidental (Nee, Colegrave et al. 2005). The mere appearance of a conserved value is insufficient for a conservation law. The key insight into identifying nontrivial

conservation laws computationally is that the candidate equations should predict relationships between dynamics of subcomponents of the system. More precisely, the conservation equation should be able to predict relationships among derivatives of groups of variables over time, derivatives that we can also readily calculate from new experimental data.

One instance of such a metric is the *partial derivatives* between *pairs* of variables. For example, in a two-dimensional system we could measure variables  $x(t)$  and  $y(t)$  over time. The system's partial derivatives estimated from time-series data would then be  $x'/y' \approx \Delta x/\Delta y$  and  $y'/x' \approx \Delta y/\Delta x$  (where  $x'$  and  $y'$  represent the time-derivative of  $x$  and  $y$ ). Similarly, given a candidate conservation equation  $f(x,y)$ , we can derive predicted values through differentiation:  $(\delta f/\delta y) / (\delta f/\delta x) \approx \delta x/\delta y$  and  $(\delta f/\delta x) / (\delta f/\delta y) \approx \delta y/\delta x$ . We can now compare  $\Delta x/\Delta y$  estimates from the experimental data with  $\delta x/\delta y$  predictions from a candidate conservation expression  $f(x,y)$  to measure how well it predicts intrinsic relationships in the system. In higher dimensional systems, multiple variable pairings and higher order derivatives yield a plethora of criteria to use. The section "Calculating the Predictive Ability" below details how to take accurate partial derivatives of  $f$  as it must be a symbolic derivative with inter-variable dependencies for higher-dimensional systems. Using the partial derivative pairs, we define a new type of search criteria for measuring how well a candidate analytical expression represents a nontrivial invariance over the experimental data.



**Figure 11.2.** The computational approach for detecting conservation laws from experimentally collected data. (A) First, calculate partial derivatives between variables from the data, then search for equations that may describe a physical invariance. To measure how well an equation describes an invariance, derive the same partial derivatives symbolically to compare with the data. Finally, return the most parsimonious equations for the hypothesized physical laws. (B) The representation of a symbolic equation in computer memory is a list of successive mathematical operations. (C) This list representation corresponds to a graph, where nodes represent mathematical building blocks and leaves represent parameters and system variables. Both (B) and (C) correspond to the equation  $f(\theta,\omega)=17.719-4.771\omega^2+4.714\cos(\theta)-\omega^2\cos(\theta)$ . To search for conservation equations, the algorithm mutates and recombines these structures to search the space of equations.

An important consequence of the partial derivative pair measure is that it can also identify relationships that represent other nontrivial identities of the system beyond invariants and conservation laws. For example, if the system is confined to a manifold, the manifold equation can also derive accurate partial derivative pairs. Similarly, the partial derivative pair can identify equations such as Lagrangian equations, the energy equivalent to the *equation of motion* in classical mechanics, which summarizes the systems dynamics, but is not invariant.

One can control, to an extent, the type of law that the system might find by choosing what variables to provide to the algorithm. For example, if we only provide position coordinates, the algorithm is forced to detect a manifold in the system's state-space. If we provide velocities, the algorithm is biased to find energy laws. If we additionally supply accelerations, the algorithm is biased to find force identities and equations-of-motion. There may exist, however, other or previously unknown analytical laws given these or other types of variables.

## **Results**

We used the algorithm summarized in Figure 11.2 to search for analytical laws in data captured from several synthetic and physical systems using various sets of system variables. We present here key results for a number of physical experimental systems; See section "Detecting Laws in Synthetic Systems" below for a study of synthetic systems, geometric symmetries, and manifolds. A video of these systems and visualizations of the search for their law expressions is available online (Schmidt and Lipson 2009).

We collected data from standard experimental systems typically used in undergraduate physics education: An air-track oscillator and a double pendulum (Figure 11.3). After

placing infrared markers on the moving components, we placed the target system in an arbitrary initial condition and recorded its transient behavior using cameras and motion-tracking software. This process provided time-series data of the marker positions. We then processed the numerical derivative of the positions to obtain velocities, accelerations, and so forth.


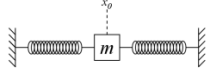
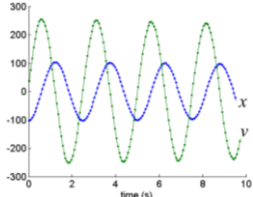
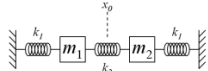
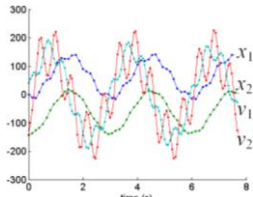

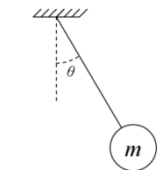
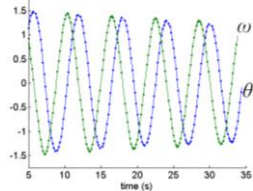
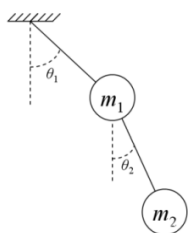
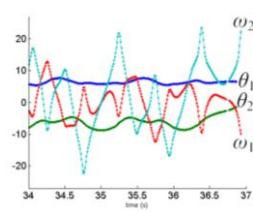
Without any additional information, system models, or theoretical knowledge, the search using the partial derivative pairs criterion was able to find several analytical law expressions directly from this data. We experimented on two configurations of the air-track: two-spring single-mass, and three-spring double-mass. Similarly, we collected time-series data from a pendulum and a double-pendulum (Figure 11.3) using motion-tracking.

The single-car air-track is a harmonic oscillator with slight damping from the air and its two springs. With only minimal noise and damping, it was the simplest physical system that we examined. Given velocity and position data from 30 seconds of observation, the algorithm detected the system's energy conservation and Lagrangian equations within five minutes. Given additionally acceleration data, it detected the system's differential equation of motion corresponding to Newton's second law.

The double-mass air-track consisted of two coupled harmonic oscillators of different masses. There was significant noise in this dataset as a result of compression of the middle spring. The algorithm still detected the Lagrangian and Hamiltonian equations.

The pendulum is a nonlinear oscillator. Given only position data, the algorithm detected that the device is confined to a circle. Given angular positions, velocities, and accelerations, it detected energy conservation, the Lagrangian, and the Newtonian equation of motion. The algorithm also detected several inexact expressions through



Physical System	Schematic	Experimental Data	Inferred Laws
			$114.28v^2 + 692.32x^2$ <b>Hamiltonian</b> $v^2 - 6.04x^2$ <b>Lagrangian</b> $a - 0.008v - 6.02x$ <b>Equation of motion</b>
			$-142.19x_1 - 74.65x_2 + 0.12x_1^2 -$ $1.89x_1x_2 - 1.51x_2^2 - 0.49v_2^2 +$ $0.41v_1v_2 - 0.082v_1^2$ <b>Lagrangian</b>
			$1.37 \cdot \omega^2 + 3.29 \cdot \cos(\theta)$ <b>Lagrangian</b> $2.71\alpha + 0.054\omega - 3.54\sin(\theta)$ <b>Equation of motion</b> $(x - 77.72)^2 + (y - 106.48)^2$ <b>Circular manifold</b>
			$\omega_1^2 + 0.32\omega_2^2 -$ $124.13\cos(\theta_1) - 46.82\cos(\theta_2) +$ $0.82\omega_1\omega_2\cos(\theta_1 - \theta_2)$ <b>Hamiltonian</b>

**Figure 11.3. Summary of laws inferred from experimental data collected from physical systems. Depending on the types of variables provided to the algorithm, it detects different types of laws. Given solely position information, the algorithm detects position manifolds; given velocities the algorithm detects energy laws; given accelerations, it detects equations of motion and sum of forces laws. These laws contain bulk parameters.**

small angle approximations – for example using  $x$  in place of  $\sin(x)$  and  $-x^2$  in place of  $\cos(x)$ . To detect the complete nonlinear trigonometric terms, the algorithm required data spanning larger angles (roughly  $\pm 40^\circ$ ).

The double-pendulum is the most complex system we studied. It is a coupled nonlinear oscillator system that exhibits rich dynamics (Jaeckel 1998) and chaos at certain energies (Shinbrot, Grebogi et al. 1992) making it challenging to model (Mor

M 2007; Liang and Feeny 2008). We focused only on detecting its energy laws. Similar to the single-pendulum, there are several approximate equations that mask the identification of its exact laws. Additionally, there is significantly higher measurement noise and dampening errors due to higher velocities of the second arm. However, these challenges were overcome by balancing data measured from the double pendulum while operating at its two different regimes – namely, in-phase and chaotic regimes.

An interesting approximate law for the double pendulum that emerged was conservation of angular momentum. Given only data measured while the pendulum was chaotic (e.g. at high energy), the algorithm tends to fixate on this law. The conservation of momentum equation is simpler than other valid laws and is approximately correct for high velocities where gravity is negligible, as with the high energy chaotic dataset.

Similarly, given only data from low velocity in-phase oscillations, the algorithm fixated on small angle approximations and uncoupled energy terms. By combining the chaotic data with low velocity in-phase oscillation data, the algorithm detected the precise energy laws.

### **Performance**

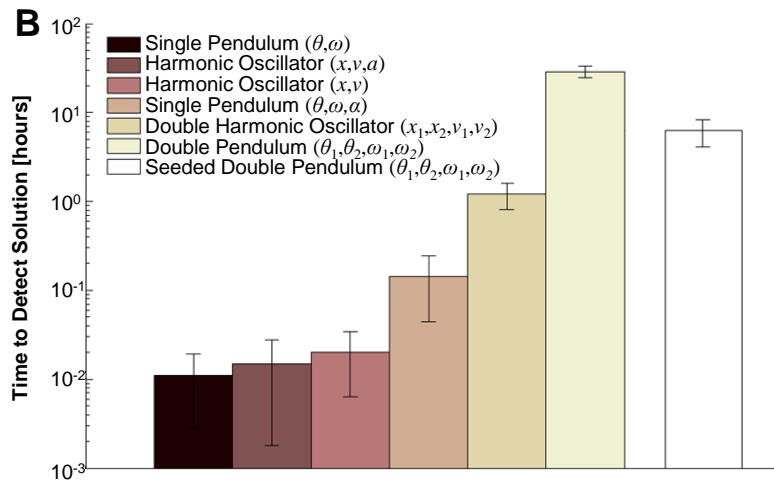
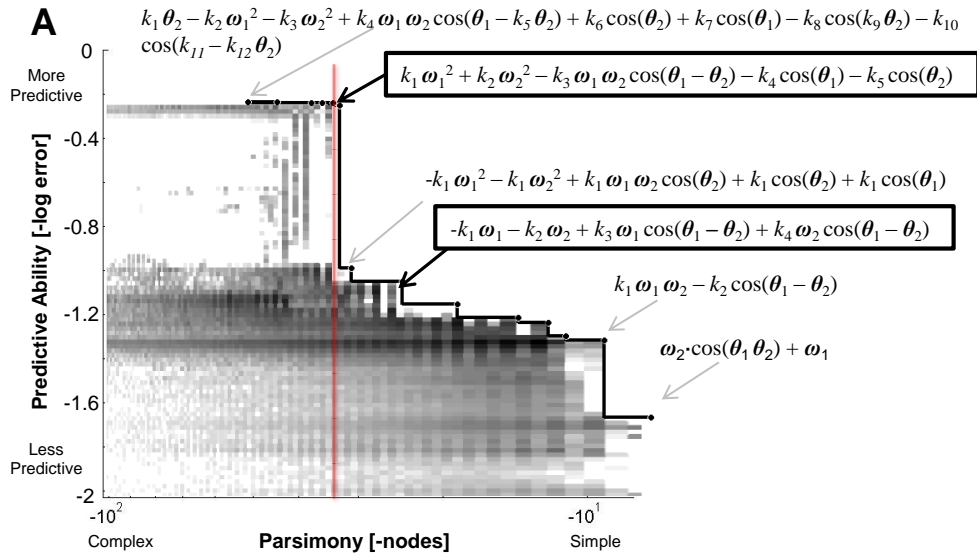
Any “good” scientific theory must be both predictive and parsimonious. Similarly, a key challenge of any machine learning algorithm is balancing accuracy versus parsimony. Some equations may be more accurate but *overfit* the data, while others may be more parsimonious but oversimplify (Edwin and Jordan 2003; Gregory, Denis et al. 2003); the right balance is difficult to specify in advance. Instead of producing a single result, the algorithm produces a small set of final candidate analytical expressions on the accuracy-parsimony *Pareto front*, which represents the tradeoff

between equation complexity and the predictive ability on the experimental data. We measured parsimony as the inverse of the number of terms in the expression.

The Pareto front for the double pendulum (Figure 11.4A) reveals a few particularly simple equations that predict the partial derivative pairs very accurately. Predictive accuracy was measured using cross-validation with the partial derivative pairs criterion. Numerically, the nature of the partial derivative pairs criterion tends to produce a large inflection where predictive ability jumps rapidly at some minimum complexity. Predictive ability then improves only marginally with more complex equations (Figure 11.4A). It is interesting to note that the conservation of angular momentum equation lies on the Pareto front, though it is inexact. The double pendulum's Hamiltonian lies at the inflection. In all of our experiments, the solution at this inflection has been an exact theoretical law.

Searching a space of equations for a natural law and discovering the Pareto front can be a computationally intensive task, possibly requiring several hours or days of computation. However, the search over function-space is readily parallelizable as many candidate functions need to be evaluated simultaneously. We distributed our computations over eight quad-core computers using the island-population model (Christian, Marc et al. 2003; Francisco, Giandomenico et al. 2005).

A 32-core implementation detected two-dimensional geometric invariants in approximately 5 minutes. The single-mass air-track laws take approximately 10 minutes. The double-mass air-track laws take approximately one to two hours. The pendulum laws take approximately 15 minutes. And the most challenging double-pendulum system takes approximately one to two days of computation (Figure 11.4B).



**Figure 11.4. Parsimony vs. accuracy, and performance. (A)** The Pareto front (solid black curve) for physical laws of the double-pendulum and the frequency of sampling during the invariant equation search (grayscale). The Pareto front shows the trade-off between equation complexity (or parsimony) and ability to model a predictive invariance. At a critical complexity of  $\sim 32$ , there is a strong point of inflection. The equation at the inflection corresponds to the exact energy conservation law of the double-pendulum, highlighted. A second momentum conservation law encountered is also highlighted. **(B)** The computation time required to detect different physical laws for several systems. The computation time increases with the dimensionality, equation complexity, and noise. A notable exception is the bootstrapped double pendulum, where reuse of terms from simpler systems helped reduce computational cost by almost an order of magnitude, suggesting a mechanism for scaling higher complexities.

In the worst case, the time to identify the equations depends exponentially on the complexity of the expression itself and roughly quadratically on the system dimensionality (Figure 11.4B). The impact of noise also couples with these factors. For comparison, the simulated double-mass air-track and simulated double-pendulum datasets (where measurements are noiseless) take approximately one-tenth of the computational effort to analyze. A summary of performance versus noise level is provided in the section "Impact of Noise" below.

### **The Justification Problem**

Though the algorithm can detect physical laws in their mathematical form, we are still faced with the challenge of justifying and giving words to their meaning. One difficulty is that we cannot know with certainty the units of bulk constants in the law expressions – for example combinations of masses, lengths, etc. embodied in the system. Secondly, the equation may model something that is inherently difficult to observe directly, such as total energy.

A more systematic approach to parsing the coefficients is to analyze multiple datasets from the same systems, albeit with different configurations and parameters. To demonstrate this approach, we used several virtual double-pendula with randomly chosen masses and lengths, to generate several new synthetic datasets. We fit the free coefficients of the automatically-discovered model to each dataset, and then invoked the equation search algorithm again to seek a relationship between the coefficients and the parameter sets. Arbitrarily setting  $k_1=1$ , the algorithm identified that  $k_2=m_2L_2^2/(m_1L_1^2 + m_2L_1^2)$ ,  $k_3=2m_2L_2/(m_1L_1 + m_2L_1)$ ,  $k_3=19.6/L_1$ , and  $k_4=19.6m_2L_2/(m_2L_1^2 + m_1L_1^2)$  where 19.6 is the only absolute constant whose units are necessarily  $m/s^2$ . A similar approach can be used to identify coefficients that vary slowly over time, for example due to damping, creeping, or ecological drift. In such

cases, the multiple datasets would come from different time windows of the same system.

### **Bootstrapping**

Thus far, the algorithm has detected natural laws *ab initio* without prior knowledge about physics, kinematics, or geometry, with a growing performance cost for increasingly complex systems. In contrast, scientists are able to leverage knowledge from simpler systems to explain more complex systems. Can an algorithm do this as well?

One method to utilize prior knowledge is *seeding* the equation search by initializing the algorithm's initial set of candidate expressions with terms from equations from simpler systems. For example, the single-pendulum (nonlinear oscillation) and the double-harmonic oscillator (coupled oscillation) equations provide clues to the laws governing the more complex double-pendulum (coupled nonlinear oscillation). To seed the set of equations for analyzing the double-pendulum, we shuffled terms of the simpler systems, exchanging velocity symbols with double-pendulum velocity variables, etc., and randomized parameters to generate many inexact initial expressions. This seeding approach does not constrain the equation search, but simply biases it to reuse terms from previous laws.

Bootstrapping the double-pendulum search with the single-pendulum and double-harmonic oscillator terms reduced the search time by nearly an order of magnitude, from 30-40 hours of computation to 7-8 hours (Figure 11.4B). Based on this result, we conjecture that bootstrapping may be critical for detecting laws in higher order systems that are veiled in complexity. We also expect there are more effective means to utilizing prior information, including human expert knowledge.

A statistical analysis of the sub-expression frequency and complexity across populations of various physical systems revealed that terms that appear more frequently than expected for their complexity tend to be more physically meaningful, such as trigonometric terms representing potential energies, squared velocities representing kinetic energies, or linear force combinations. These terms may comprise an "emergent alphabet" for describing a range of systems, which could accelerate their modeling and simplify their conceptual understanding.

### **Conclusions**

In conclusion, we have demonstrated the automatic discovery of physical laws, from scratch, directly from experimentally-captured data. The presented approach detected nonlinear energy conservation laws, Newtonian force laws, geometric invariants, and system manifolds in various synthetic and physically implemented systems without prior knowledge about physics, kinematics or geometry. The concise analytical expressions found are amenable to human interpretation and help reveal the physics underlying the observed phenomenon.

Might this process diminish the role of future scientists? Quite the contrary. Scientists may use processes such as this to help focus on interesting phenomena more rapidly, and interpret their meaning. Much like design automation allows engineers to delegate mundane design tasks to computers and focus more on creative and conceptual issues, automated mining processes might elevate scientists to think of new conceptual frameworks, leaving machines to see if these new frameworks help generate more predictive and parsimonious explanations to observed phenomena.

## Materials and Methods

### *The Predictive Ability Criterion*

To search for potential conservation equations, we need a method that discriminates trivial equations, such as coincidental invariants, from equations that represent intrinsic relationships, such as energy conservation. We define a potential invariant equation to be *nontrivial* if it can predict differential relationships between two or more variables.

One such relationship that is readily quantifiable from both the equation and experimental data is the partial derivative between pairs of variables. If our experiments collect time-series data, we can estimate the partial derivative between any pair of variables by taking the ratio of their numerical derivatives over time. For example, in a system with two state-variables  $x$  and  $y$ :

$$\frac{\Delta x}{\Delta y} \approx \frac{dx}{dt} / \frac{dy}{dt} \quad (\text{Equation 11.1})$$

We use nonparametric fitting – local polynomial fits (Cleveland and Devlin 1988) – to estimate the time-derivatives of each state-variable. In the case where we do not have time-series data, but instead random point samples, we could alternatively estimate the partial derivatives directly using two-dimensional non-parametric fitting.

A candidate equation – an equation we wish to test for triviality – can also derive the same partial derivatives between variable pairs using basic calculus. We do this by taking the ratio between partial derivatives of the equation. For example, for an equation  $f(x,y)$  over variables  $x$  and  $y$ :



$$\frac{\delta x}{\delta y} = \frac{\delta f}{\delta y} / \frac{\delta f}{\delta x} \quad (\text{Equation 11.2})$$

We now have two estimates of the partial derivative: one estimated from the data, and one predicted by the candidate equation  $f$ . To measure how well the equation predicted this relationship, we take the difference of (Equation 11.1) and (Equation 11.2) over the dataset.

$$-\frac{1}{N} \sum_{i=1}^N \log \left( 1 + \text{abs} \left( \frac{\Delta x_i}{\Delta y_i} - \frac{\delta x_i}{\delta y_i} \right) \right) \quad (\text{Equation 11.3})$$

There are many metrics for combining the residuals – such as squared-error, mean error, correlation, etc. Here, we chose to use the *mean-log-error* for numerical reasons. The magnitude of the partial derivatives can grow large when the denominator approaches or crosses zero. The mean log-error squashes these high-magnitude residuals, while not discarding them entirely. In cases where the denominator is precisely zero, we discard the data sample. By convention, we measure the negative mean-log-error to define a maximization criterion.

### ***Calculating the Predictive Ability***

Here we detail the predictive ability calculation in greater generality. While Eqns. (Equation 11.1) and (Equation 11.2) work for 2-dimensional systems using only numerical approximations, we need to consider symbolic relationships for higher order systems.

Specifically, we need to handle the case where one variable is dependent on another in order to calculate partial derivatives in (Equation 11.2) correctly. Consider calculating

$\delta x/\delta y$  in a 3-dimensional system with variables  $x$ ,  $y$ , and  $z$ . When taking the partial derivative of  $f(x,y,z)$ , we can't assume variable independence in general. Therefore, we need to perform a symbolic derivative.

For example, consider the equation of a sphere:  $f(x,y,z) = x^2 + y^2 + z^2$ . When calculating  $\delta f/\delta x$ , we must consider  $y$  and  $z$  being dependent on  $x$  or vice-versa. Using the chain-rule, the symbolic derivative is thus:

$$\frac{\delta}{\delta x} [x^2 + y^2 + z^2] = 2x + 2y \frac{\delta y}{\delta x} + 2z \frac{\delta z}{\delta x} \quad (\text{Equation 11.4})$$

In order to evaluate  $\delta f/\delta x$  we need to fill in the partial derivatives on the right-hand-side of (Equation 11.4). We have already approximated these values from the data in (Equation 11.1). Therefore, we can re-write (Equation 11.4) as:

$$\frac{\delta}{\delta x} [x^2 + y^2 + z^2] \approx 2x + 2y \frac{\Delta y}{\Delta x} + 2z \frac{\Delta z}{\Delta x} \quad (\text{Equation 11.5})$$

In general however, we should *not* assume that *every* variable is interdependent on all others – only a subset. For example in a 3-dimensional system, we only need to assume one pair of dependent variables; and in a 4-dimensional system, two pairs. So, continuing this example of the sphere equation, we have either:

$$\frac{\delta}{\delta x} [x^2 + y^2 + z^2] \approx 2x + 2y \frac{\Delta y}{\Delta x} \quad (\text{Equation 11.6})$$

or

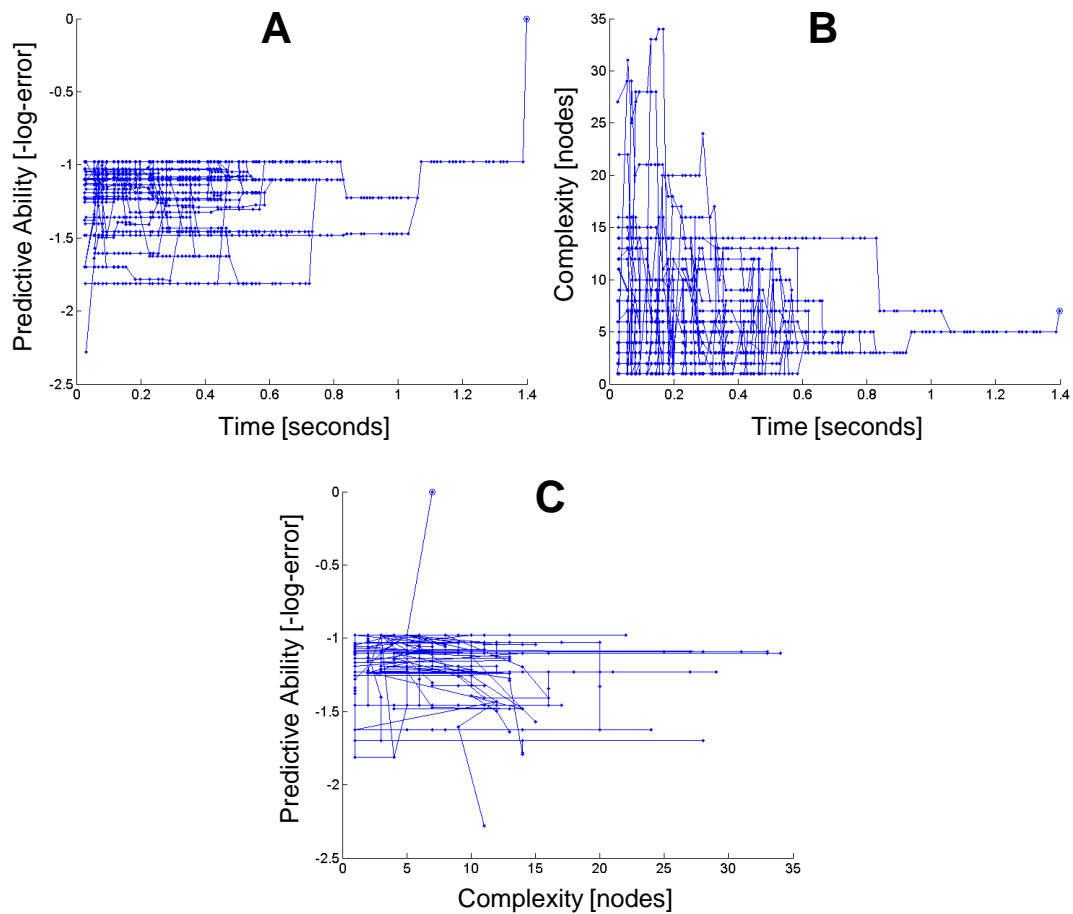
$$\frac{\delta}{\delta x} [x^2 + y^2 + z^2] \approx 2x + 2z \frac{\Delta z}{\Delta x} \quad (\text{Equation 11.7})$$

For the general case, we can pick either case (Equation 11.6) or (Equation 11.7) for our calculation of (Equation 11.3). We call this choice the *variable pairing* – which variables we assume are interdependent. We now refine (Equation 11.3) – the measure of predictive ability – to incorporate the variable pairing:

$$\min_{\text{pairing}} \left\{ -\frac{1}{N} \sum_{i=1}^N \log \left( 1 + \text{abs} \left( \frac{\Delta x_i}{\Delta y_i} - \frac{\delta x_i}{\delta y_i} \Big|_{\text{pairing}} \right) \right) \right\} \quad (\text{Equation 11.8})$$

We could optionally measure error using all possible pairings. However, we have found empirically that taking the worst-case pairing, as in (Equation 11.8), provides the best results for our computational invariant equation search.

One final adjustment we can make to the partial derivative pair metric is the sign of the of the  $\Delta x/\Delta y$  and  $\delta x/\delta y$  terms in (Equation 11.8). The partial derivative pairs define a cloud of line segments in phase space, therefore we are only interested in matching the line but not necessarily the direction of the line. Negating the  $\Delta x/\Delta y$  term or taking the absolute value of both can affect the signs of terms in the optimal equation (for example, sign differences between Lagrangian and Hamiltonian equations).



**Figure 11.5. Ancestor trajectories in equation space while searching for the equation of an ellipse. Dots indicate crossover and mutation events while lines represent parameter tuning over time. (A) Several initially random equations with varying predictive ability evolve independently before coalescing toward the exact solution over the running time of the algorithm. (B) The ancestors also vary in equation complexity – measured as the number of nodes in their expression trees. Initial equations tend to have higher complexity, but simplify over time toward the exact solution. (C) The same trajectories plotted over predictive ability and complexity shows the ancestor trajectories converge toward a simple and high predictive ability neighborhood before finding the correct equation structure whose parameters can be tuned to the exact solution.**

### *Searching the Space of Implicit Equations*

The partial derivative pairs metric, (Equation 11.3), effectively defines a landscape over the space of equations. While the landscape is difficult to visualize due to its dimensionality and size, it is smoother and more well-defined than one might expect.

Accuracy	Equations in Sequence	Event
<b>-1.4197</b>	$x + x - c_3 - y$	<i>random</i>
<b>-1.41347</b>	$x + x + x - c_4 - y$	<i>mutation</i>
<b>-1.41339</b>	$x + x + x - \sin(c_3) - y$	<i>mutation</i>
<b>-1.13805</b>	$x + x + x - \sin(y) - (x - x)$	<i>crossover</i>
<b>-1.08904</b>	$(x + x) \cdot x - \sin(y) - (x - x)$	<i>mutation</i>
<b>-1.08574</b>	$(x + x) \cdot x - \sin(y) - c_1$	<i>mutation</i>
<b>-1.01841</b>	$(x + x) \cdot x - y - c_1$	<i>mutation</i>
<b>-0.978484</b>	$(x + x + x) \cdot x - y - c_{13}$	<i>mutation</i>
<b>-0.914336</b>	$(x + y - c_3) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
<b>-0.303559</b>	$(x + y - c_4) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
<b>-0.0692607</b>	$(x + y - \sin(x)) \cdot y + x \cdot x \cdot c_{15}$	<i>crossover</i>
<b>-0.0140815</b>	$(x + y - x) \cdot y + x \cdot x \cdot c_{15}$	<i>mutation</i>
<b>-0.0050732</b>	$(x + y - x) \cdot y + x \cdot x \cdot c_{16}$	<i>mutation</i>
<b>-0.0050732</b>	$y \cdot y + c_3 \cdot x \cdot x$	<i>mutation</i>

**Figure 11.6. Sequence of solutions as they evolve to model the equation of an ellipse. This sequence represents a single trajectory in Figure 11.5. Small mutations and crossover events during the evolutionary search slowly converge this sequence onto the exact equation.**

Our method uses genetic programming to explore this landscape. In fact, most of the time, starting from a small number of random initial points in the landscape, this method can descend to the global optimal equation. We call the paths the algorithm takes to the final solution its trajectory in equation space.

See the description in the section "Symbolic Regression," on page 4 for a general description and background of the symbolic regression problem.

One way to visualize the evolution of the equation genome is to track the ancestors of the final equation over the running time of the algorithm. Figure 11.5 shows the ancestry trees for the equation of the ellipse. Several initially random equations evolve independently before coalescing. Predictive ability is initially low and some ancestors

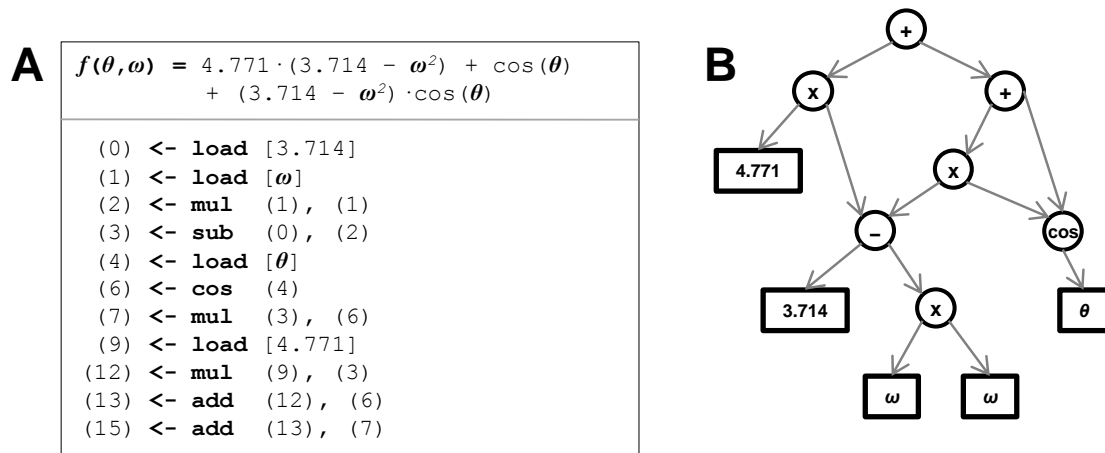
parent less accurate equations that eventually lead toward the exact solution (Figure 11.5A). Equation complexity is also initially high on average (Figure 11.5B). After several generations however, the ancestry converges to simple and predictive equations, eventually finding an equation whose parameters can be tuned to find the exact solution (Figure 11.5C).

We can also look at an individual trajectory (Figure 11.6) to see how the equations vary during the evolutionary search. The first equation is randomly initialized and has poor accuracy. Gradually, point mutations vary individual terms in the equation. Crossovers introduce larger changes, such as adding or replacing terms evolved in other ancestry sequences. In each step, the accuracy improves, until convergence onto the exact ellipse equation.

### ***Representing Invariant Equations***

The acyclic graph (Figure 11.7B) represents symbolic equations and is encoded internally as floating-point assembly. Operations can load an input variable or a parameter value, or perform a floating-point operation on any previous operation outputs (e.g. *add*, *subtract*, *multiply*, *sine*, or *cosine* commands). Each operation represents a leaf or parent node in the acyclic graph. The graph is rooted by the final operation in the list. Figure 11.7A shows a raw encoding of an example equation.

We can construct the graph of a list encoding by tracing backward from the last operation recursively. One notable consequence of this encoding is that some operations are unconnected in the graph – no operations branching from the output node may reference certain nodes. In effect, these vestigial sections are free to drift during regression since they have no impact on the equation (phenotype). These sections are omitted in Figure 11.7A.



**Figure 11.7. Two equivalent representations of an example equation  $f(\theta, \omega) = 17.719 - 4.771 \cdot \omega^2 + 4.714 \cdot \cos \theta - \omega^2 \cdot \cos \theta$ . (A) The algorithm stores and evolves equations represented by a list of floating point operators over a system's variables. Each operation can load a variable, load a parameter, or perform an mathematical operation on any previous operation. Unused lines have been omitted for clarity. (B) The raw list can be interpreted more intuitively by an acyclic graph where several sub-trees are reused by multiple terms. Both (A) and (B) represent the same equation.**

We initialize the algorithm with random equations by generating a random list of floating-point operations, limited to 128 operations. This puts a deep limit on the size of the equation graph, and narrows the search to human-interpretable equations (equations we could fit on a piece of paper). Each node could represent one of five types of mathematical operations, two to four variables, or a parameter constant. Ignoring the infinite parameter space, this is effectively a search space of roughly  $10^{108}$  parameterized equations.

## Analysis of Results

### *Detecting Laws in Synthetic Systems*

In addition to physical laws such as Hamiltonians, Lagrangians, and equations of motion, the partial derivative pair criterion can also decipher implicit equations and geometric constraints. Table 11.1 summarizes the algorithm's search over time and the

Pareto fronts for several synthetic manifolds and simulated dynamical systems.

Systems with parameter constants tend to exhibit gradual convergence whereas parameter-less equations converge rapidly at differing times. There is a similar inflection trend among all the Pareto fronts – an equation with some minimum complexity achieves very high predictive ability. The inflection of the double linear oscillator is more subtle, which we suspect is due to the large number of terms and polynomial approximations in its Hamiltonian equation.

The algorithm's search over a space of equations for a natural law and building the Pareto front is a computationally intensive task, possibly requiring several hours or days of computation. However, the search is readily parallelizable as many candidate functions need to be evaluated simultaneously. We distributed our computations using the island-population model (Christian, Marc et al. 2003; Francisco, Giandomenico et al. 2005) and used a fitness-prediction model (Schmidt and Lipson 2008) to reduce overall computational cost and to improve the local search gradient.

In a 32-core implementation, 10 minutes for the pendulum to a day for the double pendulum. The time for two-dimensional geometric invariants to be found on the Pareto front during the algorithm's search was approximately 5 minutes. The single-mass air-track laws took approximately 10 minutes to appear. The double-mass air-track laws took approximately one to two hours to appear. The pendulum laws took approximately 15 minutes to appear. And the most challenging, the double-pendulum system, took approximately one to two days of computation.



**Table 11.1.** The predictive ability and Pareto fronts of several synthetic manifolds and simulated dynamical systems. Error bars denote the standard error of predictive ability

System	Predictive Ability Over Time	Accuracy/Complexity Pareto Front
<b>Circle:</b> $x^2 + y^2$		
<b>Elliptic Curve:</b> $x^3 + x - y^2$		
<b>Sphere:</b> $x^2 + y^2 + z^2$		
<b>Linear Oscillator:</b> $a - 0.1 \cdot v + 3 \cdot x$		
<b>Linear Oscillator:</b> $x^2 + 0.3 \cdot v^2$		

**Table 11.1 (cont.) The predictive ability and Pareto fronts of several synthetic manifolds and simulated dynamical systems. Error bars denote the standard error of predictive ability.**

System	Predictive Ability Over Time	Accuracy/Complexity Pareto Front
<b>Pendulum:</b> $\alpha - 9.8 \cdot \sin(\theta)$		
<b>Pendulum:</b> $\omega^2 - 9.8 \cdot \cos(\theta)$		
<b>Double Linear Oscillator</b> $x_1^2 + (x_1 - x_2)^2 + (1 - x_2)^2 + 2 \cdot v_1^2 + v_2^2$		
<b>Double Pendulum</b> $\omega_1^2 + 0.5 \cdot \omega_2^2 + \omega_1 \omega_2 \cos(\theta_1 - \theta_2) - 19.6 \cos(\theta_1) - 9.8 \cos(\theta_2)$		

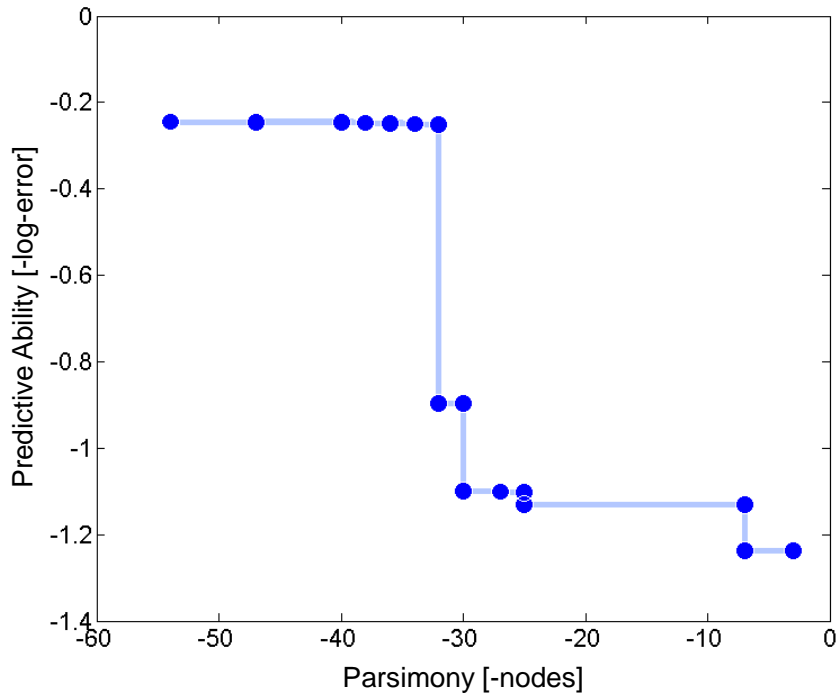
### ***Equation Accuracy and Complexity Tradeoff***

For any finite set of experimental data, there is potentially an infinite set of equations that maximize any type of error metric. For example, a 1000<sup>th</sup> order polynomial can perfectly fit any dataset of 1000 or fewer unique data points. While it is immensely more difficult to find arbitrarily accurate equations using the partial derivative predictive ability criterion, it is still important to have some qualitative understanding of what the domain of equations looks like.

Consider the relationship between equation complexity and accuracy of fitting the experimental data. Qualitatively there two extremes: complex equations (e.g. a Taylor series, neural networks, or Fourier series) with arbitrarily high accuracy, and the most simple models with baseline accuracy. Equations that are simultaneously simple and accuracy are the most difficult to find. Figure 11.8 shows the Pareto front of equation accuracy versus equation complexity for the double-pendulum.

The algorithm may also fail to find interesting relationships, due to either lack of convergence, inappropriate building blocks, or absence of any governing law. In this case, the front may be poorly formed with only exceedingly complex solutions reaching high predictive ability.

At certain minimum complexities, the equation's predictive ability jumps dramatically and then plateaus. We can reason this equation is the most likely candidate, as further elaborations yield marginal improvement in predictive ability. The equation at the inflection in this example is indeed the conservation of energy equation (Hamiltonian), supporting this assumption.

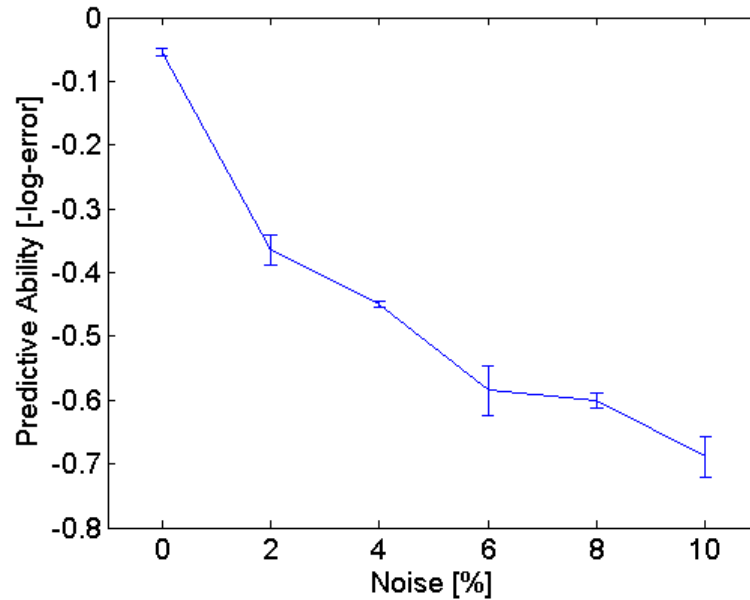


**Figure 11.8. The accuracy/complexity Pareto front of the double pendulum. The Pareto front shows the tradeoff between equation complexity and its ability to derive accurate partial derivative. At some minimum complexity (32 nodes), predictive accuracy jumps rapidly. Equations almost twice as complex improve the accuracy only marginally. These high complexity equations tend to contain the simpler exact equation, but add many smaller terms to compensate noise. The parsimonious and accurate equation at the inflection is the Hamiltonian and Lagrangian of the double pendulum.**

### *Impact of Noise*

The presence of noise can make estimating derivatives difficult because derivatives can be highly sensitive to noise. We use Loess smoothing (Cleveland and Devlin 1988) – a non-parametric fitting method – to remove high frequency noise from the motion tracking system. Loess smoothing updates each sample in the dataset by fitting a small order polynomial to the sample and its nearest neighbors.

Other methods, such as filtering and convolution, also reduce high-frequency noise, but do not readily produce estimates of the signal derivative. Using Loess smoothing, we obtain the numerical derivatives directly from the smoothing procedure by



**Figure 11.9.** The mean predictive ability on a withheld test set of the best equations detected versus the amount of normally distributed noise in the data set for the simulated double linear oscillator. Error bars show the standard error. The percent noise is the ratio of the standard deviation of the noise and the standard deviation of the original signal.

evaluating the symbolic derivatives of the local polynomial fits at each data sample.

We have examined the impact of noise on the predictive ability for the double linear oscillator (Figure 11.9). Noise reduces the ability to find accurate invariant equations substantially, either simply requiring more time to compute or obscuring the equation entirely depending on the noise strength. We measure the noise strength (percent noise) as the ratio of the standard deviation of the random noise to the standard deviation of the exact signal.

### ***Data Collection and Preprocessing***

We used motion tracking cameras and software (Vicon MX) to collect data on physical systems such as the double-pendulum. We place several infrared markers on the experimental device, place it into an arbitrary initial condition, and observe its dynamics.

The motion tracking produces time-series data of 3-dimensional Euclidean position coordinates for each infrared marker. We use many infrared markers in order to minimize noise and occlusions effects during the tracking. Afterward, we then combine the time-series of each marker to calculate the essential state-variables of the system – 2-dimensional coordinates, angles, etc. For example, in the double-pendulum, we project all 3-dimensional tracking points to its principle plane, and then calculate the angle of the two pendulum arms by taking the arctangent between segments of the infrared markers.

While motion tracking systems have become quite accurate and automated (Greg and Eric 2002), we must still handle noise and occlusion in the time-series data. Noise amplifies when the system experiences high velocities or when the number of cameras that can see a particular infrared marker changes.

In the double-pendulum, the infrared markers on the second arm become occluded from nearly all cameras when it passes behind the upper arm. In this case, the motion tracking produces null position coordinates, which we strip out before processing. Therefore, some of our time-series data contains gaps.

### ***Evolutionary Parameters***

We use the fitness prediction algorithm (Schmidt and Lipson 2005; Schmidt and Lipson 2006; Schmidt and Lipson 2008) to search over symbolic equations. The selection method was deterministic crowding selection (Mahfoud 1995), using 1% point-mutation probability and 75% crossover probability. The encoding each equation was an acyclic graph with a maximum of 128 operations/nodes (Schmidt and Lipson 2007). We used single-point crossover to exchange the operations in the parent equations. The operator set contained addition, subtraction, multiply, sine, and cosine.

**Table 11.2. Summary of Detected Approximations with Missing Building Blocks**

<b>Building Blocks</b>	<b>Detected Pendulum Law</b>	<b>Approximation Discovered</b>
*, +, -, cos(), sin()	$\omega^2 - 19.6 \cdot \cos(\theta)$	<i>Exact Solution</i>
*, +, -, sin()	$\omega^2 - 19.5999 \cdot \sin(-1.57079 + \theta)$	<i>Trigonometric identity</i>
*, +, -	$\omega^2 + 9.7108 \cdot \theta^2 - 0.7042 \cdot \theta^4$	<i>Taylor series expansion (4<sup>th</sup> order)</i>

We distributed the symbolic regression evolution over 8 quad core computers (32 total cores) using the island distributed computation method (Christian, Marc et al. 2003; Francisco, Giandomenico et al. 2005). We spread a population of 2048 equations over 32 CPU cores; therefore each island population has 64 equations.

The fitness predictor population contains 512 predictors, distributed over 32 cores. The fitness predictors consist of 128 indices into the full training data set. The predictors are evolved with deterministic crowding, using 10% mutation and 50% crossover rates.

We calculate fitness using variations of (Equation 11.8), where we modify the signs of partial derivative pairs using negation or absolute value to vary the types of equations we search for. For predicted fitness values, we only calculate (Equation 11.8) over the smaller subset of the fitness predictor rather than the entire data set.

### ***Results with Missing Building Blocks***

It is interesting to note that in the absence of appropriate building blocks, the algorithm develops approximations. For example, eliminating sine and cosine as building blocks causes the pendulum invariant to be expressed as  $\omega^2 + k_1\theta^2 - k_1\theta^4$ ,

thereby exploiting the Taylor series expansion. Eliminating cosine but not sine results in other identities, such as  $\cos(\theta) = \sin(\theta + \pi/2)$  or more complex equivalences (Table 11.2).



## CHAPTER 12. SYMBOLIC NOISE SOURCE MODELS

### **Summary**

In this chapter we propose a genetic programming approach to learning stochastic models with unsymmetrical noise distributions. Most learning algorithms try to learn from noisy data by modeling the maximum likelihood output or least squared error, assuming that noise effects average out. While this process works well for data with symmetrical noise distributions (such as Gaussian observation noise), many real-life sources of noise are not symmetrically distributed, thus this approach does not hold. We suggest improved learning can be obtained by including noise sources explicitly in the model as a stochastic element. A stochastic element is a random sub-process or latent variable of a hidden system that can propagate nonlinear noise to the observable outputs. Stochastic elements can skew and distort output features making regression of analytical models particularly difficult and error minimizing approaches inhibiting. We introduce a new method to infer the analytical model of a system by decomposing non-uniform noise observed at the outputs into uniform stochastic elements appearing symbolically inside the system. Results demonstrate the ability to regress exact analytical models where stochastic elements are embedded inside nonlinear and polynomial hidden systems.

### **Introduction**

Random noise is found in many natural and engineered systems, such as random diffusion, noisy actuators or sensors, and human input (Kulkarni 1995). Most learning algorithms handle noise by fitting the maximum likelihood or least squares error of noisy data (Kulkarni 1995; Carl Edward 1997). This approach works well when noise is distributed symmetrical about the true system output, such as white noise, Gaussian noise, and any zero mean noise superimposed over the output.

When noise exists internally in the system, it can be coupled with nonlinear components of the system. In other words symmetric internal noise can be scaled, offset, and in general transformed to produce non-symmetric noise distributions on the output. In these situations, the noise has deformed the maximum-likelihood output from the theoretical noiseless system, and the regressed models may no longer describe the analytical structure of the system.

We call this type of noise a *stochastic element* – a random process inherent to the system, affecting its behavior and observable output. Noise from stochastic elements can propagate nonlinearly to the system’s output and produce non-uniform variation.

The most common approach to handling noise is to model its expectation, either through averaging or least-squares fitting (Kulkarni 1995; Carl Edward 1997). While the expectation of a noisy system is valuable for finding a model with minimal error, it can be misleading when finding a descriptive analytical model of the system (e.g. symbolic regression). In the worst case, it can distort the observed output of the system, preventing the true system structure from being found.

In this chapter, we aim to improve regression of a noisy system based on the notion that observed noise that is coupled to the system may itself provide additional information about the system’s analytical structure. For example, if the output noise appears to grow quadratically, there is likely to be some quadratic structure in the system. Our approach is to use symbolic regression to model the output noise explicitly, decomposing noise as uniform stochastic elements inside the system to produce a noisy model. We then compare the noise observed in candidate models to the variation in the training data to calculate fitness. The final analytical model is obtained by removing the stochastic terminals used.

In the remaining sections, we discuss the distortion produced by stochastic elements, describe our approach in greater detail, show some simple results, and finish with concluding remarks.

## **Background**

### *Distortion from Stochastic Elements*

Expected values of a noisy output can disguise and distort analytical structure when the system contains internal stochastic elements (Schaffer, Ellner et al. 1986; Kleijnen 2006). Noise can be multiplied into the system or pass through a nonlinear operation to significantly change the expected output values. Figure 12.1 shows three simple examples where a stochastic element hides or distorts analytical features.

Figure 12.1a shows a sine function,  $f(x) = \sin(x)$  with a stochastic element giving rise to a random phase offset,  $f(x) = \sin(x + R)$ . The noise does not change the magnitude of the sine wave but does shift data samples left or right. The expectation of the output shows a sine function with correct phase but with smaller amplitude than the target analytical model,  $f(x) = A \cdot \sin(x)$ .

The system in Figure 12.1b is a simple linear function,  $f(x) = x$ , multiplied by a stochastic element,  $f(x) = x \cdot R$ . The multiplied noise completely hides the linear growth from the expectation. The expected output becomes simply  $f(x) = 0$ .

Figure 12.1c is a quadratic function,  $f(x) = x^2$ , with noise added to the input,  $f(x) = (x + R)^2$ . This noise again shifts the data points left or right, but does not change the y-intercept. The expected output model however is quadratic with a y-offset,  $f(x) = x^2 + A$ .

Though these are simple examples, they give insight into how stochastic elements can distort expectation models from the exact analytical model, or even hide features. In

the next section, we describe a simple approach to incorporating stochastic elements into models in order to recover exact analytical models despite this difficulty.

### ***Regressing Noisy Data***

Noise is found in almost all experimental data and is a central focus in many areas of machine learning (Arnold 2001). Here, we briefly overview how noise is traditionally handled in regression problems.

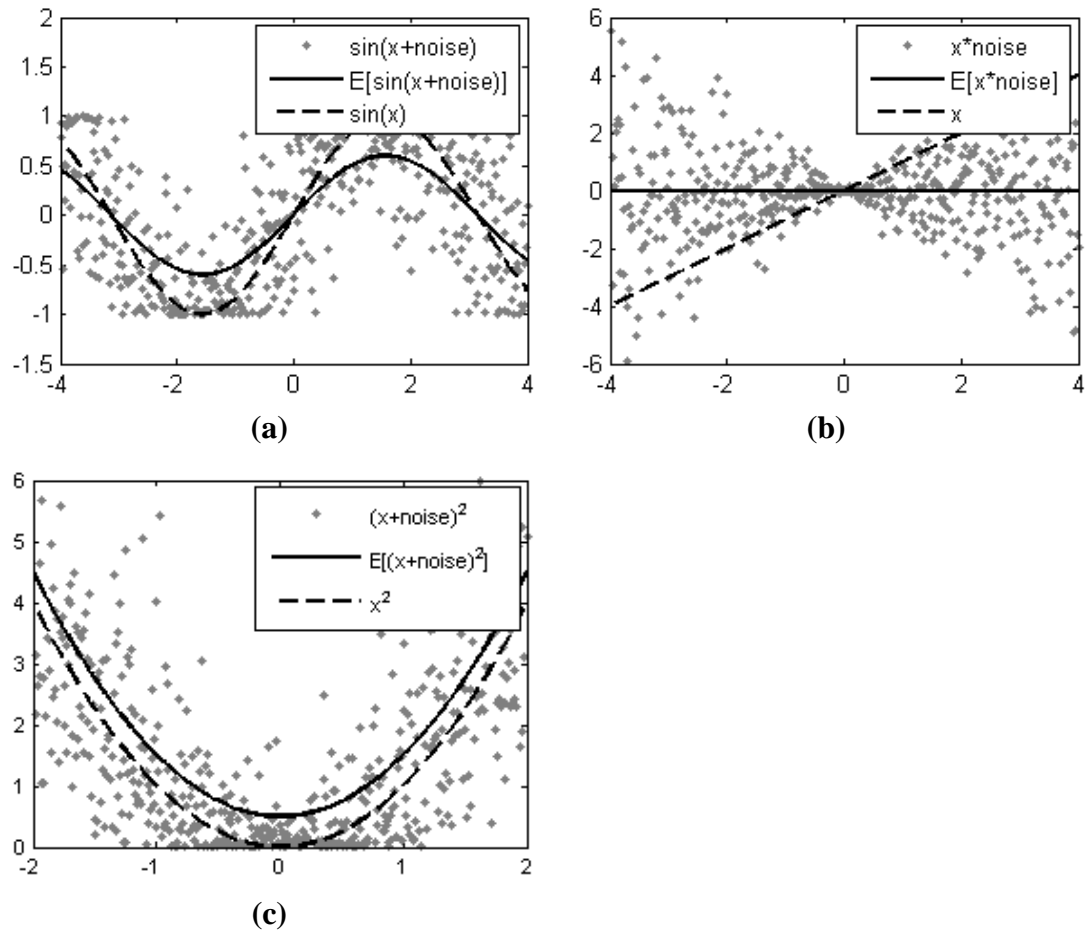
Often experimental data is pre-processed to remove outliers (Rousseeuw and Leroy 1987), remove white noise (Kleijnen 2006), and more generally, smooth features. Common techniques for preprocessing include convolving with a low-pass-filter (e.g. box or sliding window, Gaussian filter), local least-squares fitting, and spline fitting.

The aim of preprocessing is to transform the data set to be more representative of the expected outcome or maximum likelihood of the system through interpolation or statistical properties among neighboring data points. These processes make assumptions about the underlying system and its noise distribution but are still used frequently in practice to improve predictive performance.

In contrast, we are interested in exploiting the existence of nonlinear noise to reveal internal structure of the unknown system. In this sense, the goal is broader and removing noise coupled to the system could remove information.

### ***Modeling Noise and Confidence***

One is often interested in the confidence of predictions made by a regressed model. Accurate models predict the maximum-likelihood value, but the variance of outputs for this value may be large.



**Figure 12.1.** Three basic examples where a stochastic element hides or distorts analytical features of the system to different extents. Blue dots show the observed system output, the red line shows the expectation of the output, and the green line show the target analytical model with stochastic elements removed.

The most common non-parametric approach to measure confidence is to examine the residual errors of the model on the training set. This leads to a natural two-step procedure:

- (1) Regress a best fit model
- (2) Derive a statistical model of the residual error

In the case of white noise, residual errors appear uniformly distributed and can be modeled globally such as calculating its mean and variance.

If noise is coupled to the system by an internal stochastic element, the residual error may vary greatly over the input space. In this case, local statistical models are used to model confidence among neighboring inputs (Touretzky, Leen et al. 2007).

Deriving a statistical model of the residual error in this fashion requires assuming a noise distribution model, such as the normal distribution. In nonlinear regression, where an analytical model of the system is assumed, the noise distribution can be derived automatically from the model. Most commonly, confidence intervals are calculated on the model fitting parameters (Vugrin, Swiler et al. 2007). Parameter confidences then translate into nonlinear output confidence ranges on the model output.

In contrast, the method proposed in this chapter models noise explicitly in the model parametrically without a predetermined model structure.

### ***Symbolic Regression***

See the description in the section “Symbolic Regression” on page 4 for background the symbolic regression problem.

The fitness objective in symbolic regression, traditionally, is to minimize error on the training set (Koza 1992; Augusto and Barbosa 2000; Schmidt and Lipson 2005). Later in this chapter however, we define a new objective geared specifically to reward candidate solutions with noise distributions that match the noise observed in the training set.

### **Learning Noise Algorithm**

The basic idea of our approach is to include behavior of stochastic elements inside the analytical model. Instead of using an error minimization objective, we attempt to find

---

---

```
Individual ind = ( encoding E, stochastic elements S )
Input variables X
```

```
Function evaluate() :
  For each s in S
    s = random value [-1, 1]
  End
  ...
  val = evaluate ind normally
  ...
  Return val
```

---

---

```
Individual ind
Training data D of (x,y) pairs
Number of samples N
```

```
Function fitness() :
  fitness = 0
  For each d in D
    ymin, ymax
    Repeat N times
      y = ind.evaluate(d.x)
      If (y < ymin) ymin = y
      If (y > ymax) ymax = y
    End
    If (ymin < d.y < ymax)
      fitness += 1 / (ymax - ymin)
    Else
      fitness += - min(|d.y - ymax|, |d.y - ymin|)
    End
  End
  Return fitness
```

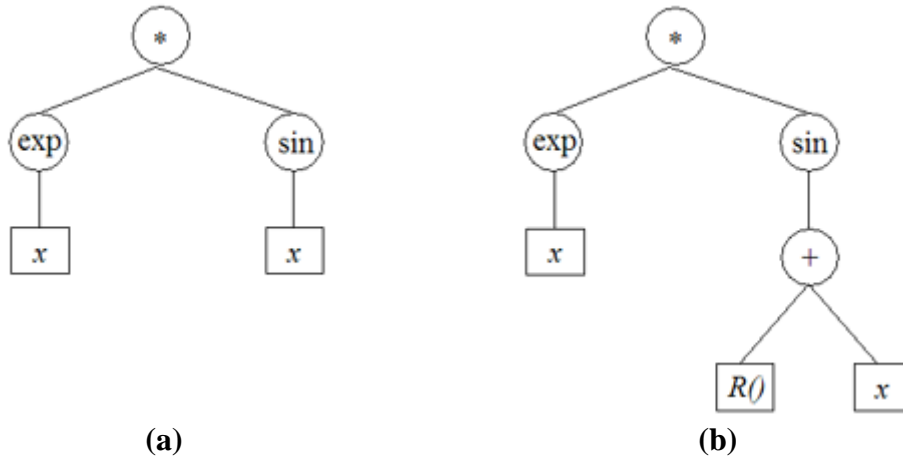
---

---

**Figure 12.2. Pseudocode for evaluating a model with stochastic noise sources to estimate the noise envelope or distribution (top), and pseudocode for calculating the resulting fitness metric for the candidate model (bottom).**

a model of stochastic elements with the simplest distribution explaining all features and noise in the training data. The final analytical model identifies the origin of noise as well as its effect on out observations.

Much research has been done on bounding noise error and modeling error



**Figure 12.3.** An example binary expression tree (a) for the function  $f(x) = e^x \sin(x)$ , and a similar tree modeling a stochastic element (b) for the function  $f(x) = e^x \sin(x) + R()$ .

distributions (Xavier and David 2003; Touretzky, Leen et al. 2007; Vugrin, Swiler et al. 2007). The distinction here is that we are modeling individual noise components explicitly inside a system. The analytical model is regressed from scratch, rather than relying on an assumed system model or distribution model.

We use symbolic regression to find an analytical model which incorporates uniform random variables to explain residual error parametrically in addition to finding a best fit. In the next two sections, we describe how we incorporate stochastic elements into candidate models and describe a new objective function to explain observed noise.

### *Decomposing Stochastic Elements*

Our basic building block for a stochastic element is a uniform random variable with range -1 to 1 inclusive that returns a random value every time it is read or evaluated by the model. Symbolic regression can incorporate this random variable anywhere in its models to help explain the noise distribution.

$$R() = \text{uniform random value } [-1, 1]$$



Nearly all types of random variables and distributions can be derived from this uniform random variable. Symbolic regression treats this variable like it would any other attribute variable, and can derive combinations and transformations to non-uniform distributions. For example, the Normal distribution can be derived from querying the uniform random variable twice:

$$Normal(0,1) = -A \cdot \sqrt{\ln(2) - \ln(R() + 1)} \cdot \cos(R() \cdot \pi)$$

Symbolic regression most commonly represents candidate solutions as expression trees (Figure 12.3a).

We treat stochastic elements as a new variable in the terminal set that can be used anywhere in the expression tree to model the noise in experimental data (Figure 12.3b). The new terminal value is special however in that it is randomized every time it is evaluated, even when appearing multiple times in the same expression tree.

### ***The Noise Distribution Objective***

Now that candidate models can include random variables, their output predictions will have some distribution. Our goal for this distribution is to explain all variation found in the training data, and do so in the narrowest and simplest way.

A distribution explains a training data point if the data point falls inside the model's distribution at that point. For example if  $f(x=10)$  has a distribution between  $[-9,-3]$ , it explains the training data point if its value is  $-6$ , but not if it is  $4$ .

We can approximate the distribution of a candidate model at a training point by sampling it. In our experiments, we find the range of output for a training input by storing the minimum and maximum output from 100 model evaluations.

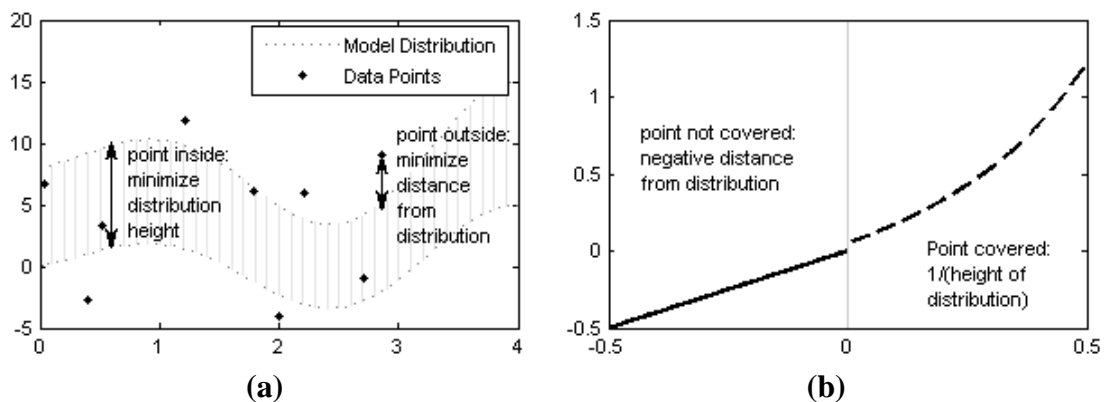
Note however that a trivial solution would be a large (or perhaps infinite) distribution where all training data lies inside the distribution. Therefore, we must introduce a second objective to minimize the size of the distribution.

If a training data point lies inside the model's distribution, we want to minimize the height of the distribution at that point. If the point is not covered by the distribution, we want to minimize the distance of that point from the distribution. We can combine these two objectives into a single fitness criterion:

$$fitness(f) = \sum_{(x,y)} \begin{cases} -\min |y - range(f(x))| & \text{if } y \notin range(f(x)) \\ 1/range(f(x)) & \text{if } y \in range(f(x)) \end{cases}$$

This is a two-step fitness objective, summarized in Figure 12.4. The model must first cover the point with its distribution, and then it must minimize the area of its distribution. As shown in Figure 12.4b, training points not explained by the distribution contribute negatively to the fitness, and points that are explained contribute positively.

Pseudocode for evaluating a model that contains stochastic elements, and for



**Figure 12.4. The fitness objective for explaining training data with a with model that has stochastic elements and output distribution. If a training point falls inside the model distribution, the objective is to minimize the height of the distribution. If the point falls outside, the objective is to minimize the distance of the point to the distribution.**

**Table 12.1. Summary of Experiment Setup**

Solution Population Size	64
Selection Method	Deterministic Crowding
P(mutation)	0.05
P(crossover)	0.75
Solution Encoding	Operation List (graph)
Operations	16
Local Variables	4
Evolved Constants	4
Inputs	1
Operator Set	$+$ , $-$ , $*$ , $/$ , $\sin$ , $\cos$
Terminal Set	$x$ , $c_1$ , $c_2$ , $c_3$ , $c_4$
Crossover	variable, single point
Fitness Sample Size	4
Distribution Samples	100

evaluating the distribution fitness of a model is shown in Figure 12.2.

### **Experiments**

We modify a symbolic regression algorithm (Schmidt and Lipson 2005) to include stochastic elements and regress based on distributions rather than error minimization. This algorithm utilizes adaptive sampling of the training set to reduce computational cost, which is particularly high for finding the output distribution of candidate models during regression.

Parameters for all experiments are summarized in Table 12.1. In deterministic crowding, offspring replace their most similar parent if they have equal or higher fitness and are discarded otherwise. Population size, mutation probability, and

crossover probability have been tuned empirically. Crossover produces a higher fit child approximately 20% of the time with these setting on the operation list encoding.

The candidate solutions (algebraic expressions) are lists of operations on local variables. The number of operations and local variables were tuned for computational performance. The encoding size, terminal set, and operator set are over-represented (no experiments requires all for convergence). Single point crossover is used on the operation list at a variable offset.

To measure fitness, the output distribution is measured on four inputs from the training set, one hundred times. The minimum and maximum values are then used to calculate the fitness described earlier.

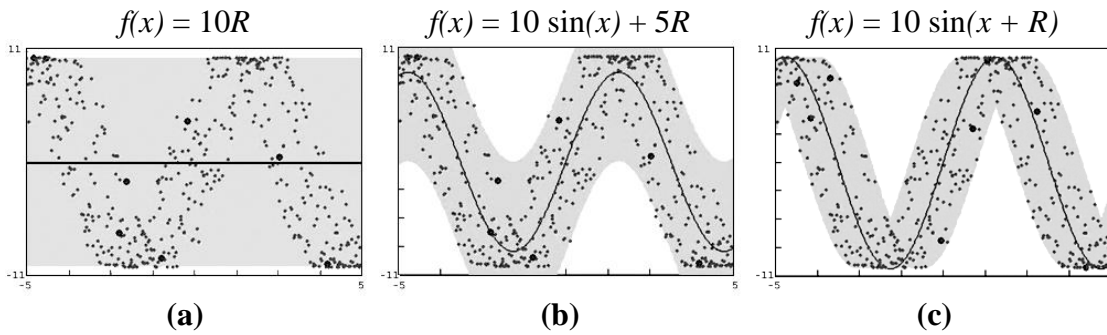
We test on three simple example systems each with a uniform stochastic element coupled in the system:

- $f_1(x) = 10 \sin(x + R)$
- $f_2(x) = x^2 \sin(x + R)$
- $f_3(x) = (x + R) - 1.5 x^3$

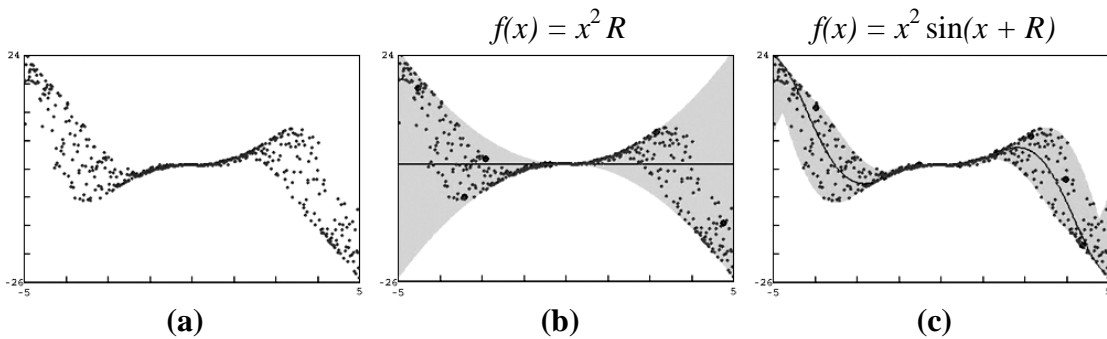
These experiments demonstrate the finding the exact structure and parameters of the system despite internal stochastic noise which offset the expected output.

## **Results**

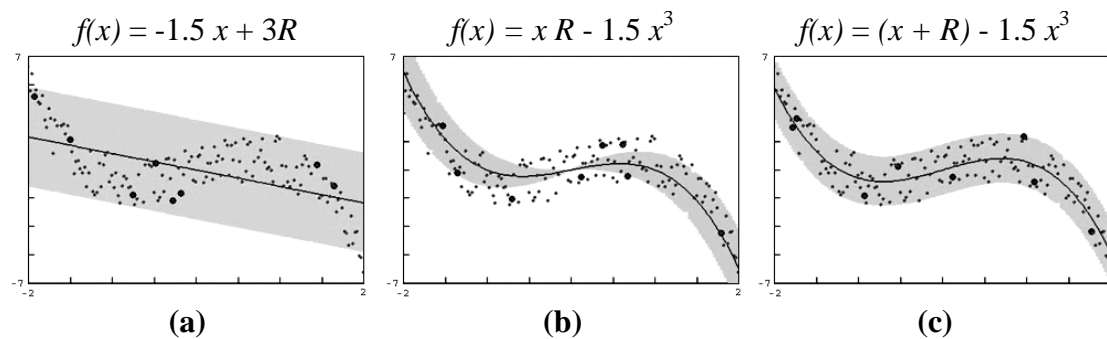
This section gives results on three simple examples of regressing stochastic elements embedded in a hidden system to demonstrate our approach. We show screen captures of different stages during regression to show the progress toward the analytical model.



**Figure 12.5.** The best model found at three points during regression of  $f(x) = 10 \sin(x + R)$ . The green points show the training data, the grey area shows the model's distribution, and the blue line shows the analytical model with stochastic elements removed.



**Figure 12.6.** The best model found at three points during regression of  $f(x) = x^2 \sin(x + R)$ . The green points indicate the training data, the grey area indicates the model's distribution, and the blue line indicates the analytical model with stochastic elements removed.



**Figure 12.7.** The best model found at three points during regression of  $f(x) = (x + R) - 1.5 x^3$ . The green points are the training data, the grey area is the model's distribution, and the blue line is the analytical model with stochastic elements removed.

The time to regress each system successfully ranged from one to five minutes. The primary computation time consists in computing the candidate model distribution at each training point. We use random sampling to determine the output ranges at each point, but a more intelligent sampling method could be used to scale the application to higher complexity systems.

Figure 12.5 shows three stages during regression of the function  $f(x) = 10 \sin(x + R)$ , where  $R$  is a stochastic element variable that returns a uniformly random number in the range  $x = -1$  to  $x = 1$  inclusive each time it is read.

Early on, candidate solutions are linear with distributions that cover all the training points – shown in Figure 12.5a. In Figure 12.5b, the solutions have inferred the sine function in the system, but the noise distribution is just added linearly to the output. In the next stage, Figure 12.5c, the solution has converged on the sine function with the stochastic element located inside the sine function.

Figure 12.6 shows the regression of the function  $f(x) = x^2 \sin(x + R)$  which is similar to the first experiment but now has a variable amplitude sine wave. Candidate solutions converge on quadratic amplitude noise very quickly – Figure 12.6b. Shortly after, the sine function is found and the analytical model converges in Figure 12.6c.

The third experiment uses a polynomial function but with noise simply added linearly to the output. This is a case where the minimum error model is the same as the analytical model but it is important that we can differentiate this type of noise as well. Figure 12.7a shows early candidate solutions are linear with an additive noise range. In Figure 12.7b, the analytical model has been found but the noise distribution has not yet explained all data points. Figure 12.7c shows the converged solution identifying the correct analytical model and its distribution.

## **Conclusions**

Stochastic elements existing inside a hidden system can produce nonlinear and non-uniform noise at the observable outputs. There are many cases where the expected value output or minimum error regression can be deceiving toward finding an exact analytical model as done in symbolic regression.

We have presented a simple approach to model stochastic elements directly as uniform random features using symbolic regression. The objective for candidate models with stochastic elements is to explain (overlap) all training data points in its distribution and minimize the area of the distribution used.

Results show this approach can find the exact analytical model despite misleading nonlinear and non-uniform output noise. In three basic experiments, regression of the output distribution found the correct system structure and location of the stochastic elements with parameters existing in the hidden system.

## CHAPTER 13. STOCHASTIC REACTION MODELS

### **Summary**

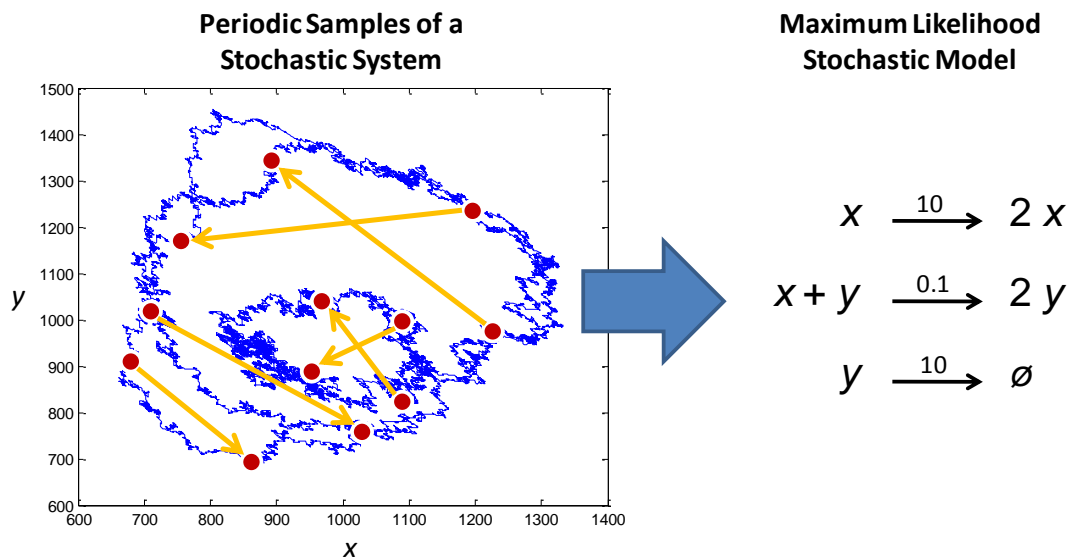
Many systems, particularly in biology and chemistry, involve the interaction of discrete quantities, such as individual elements or molecules. When the total number of elements in the system is low, the impact of individual reactions becomes non-negligible and modeling requires the simulation of exact sequences of reactions. In this chapter, we introduce an algorithm that can infer an exact stochastic reaction model based on sparse measurements of an evolving system of discrete quantities. The algorithm is based on simulating a candidate model to maximize the likelihood of the data. When the likelihood is too small to provide a search gradient, the algorithm uses the distance of the data to the model's estimated distribution. Results show that this method infers stochastic models reliably with both short time gaps between measurements of the system, and long time gaps where the system state has evolved qualitatively far between each measurement. Furthermore, the proposed metric outperforms optimizing on likelihood or distance components alone. Traits measured on the search novelty, age, and bloat suggest that this algorithm scales well to increasingly complex systems.

### **Introduction**

Stochastic systems pervade nearly all areas of science, from quantum properties of atomic particles, to chemical reactions in a chemical bath, to fluctuations in populations or ecosystems. All stochastic systems are at least partially random, making them difficult to model dynamically or deterministically. Instead, Monte Carlo methods are often employed to simulate and analyze their behavior.

A particularly important Monte Carlo method was developed by Dan Gillespie in 1977





**Figure 13.1. Overview of the modeling problem. A stochastic system evolves an exact behavior over time shown in blue. Periodically, the state of system can be measured (shown in red dots), a sample of the exact time evolution of the system. The task is to infer a maximum likelihood stochastic model (right) for this system from these periodic measurements. Actual data and solution shown.**

in order to model chemical reactions kinetics (Gillespie 1977). The Gillespie algorithm performs an exact and statistically-correct simulation of a stochastic system based on a set of discrete chemical reactions, reaction coefficients, and initial conditions. The Gillespie algorithm has been used extensively in systems biology, and also similar domains. Traditionally, the set of reactions that model a stochastic system must be developed and theorized manually by experts.

In this chapter we introduce an evolutionary algorithm that automatically hypothesizes about the reactions and reaction rates taking place in a system simply by analyzing raw experimental data, even with large time gaps between observations (see Figure 13.1). The proposed method searches over a space of reactions in order to find the maximum likelihood model that agrees with the experimental observations.

The key challenges to searching over stochastic models is the computational cost of

estimating likelihood values from a model and maintaining a search gradient. Except for only the most trivial systems, the probability density of a set of stochastic reactions cannot be solved over time. Instead, the model can be simulated (or sampled) repeatedly. However, efficient sampling methods fail over large time spans (Gillespie 2007), making it difficult to estimate distribution tails.

The proposed method overcomes this difficulty by using a two-component optimization metric. The metric attempts to maximize the log-likelihood of the data given a candidate model. However, if the likelihood is too small to provide a gradient for the search, the criterion changes to the distance of each data point to the estimated probability density of the candidate model. In effect, this distance component allows even extremely inaccurate models to improve despite having zero likelihood. Once models get close enough to the data, where their likelihoods can be estimated accurately through sampling, the metric switches to maximize the likelihood.

This metric also reduces the computational complexity, as the accuracy of estimating the tails of distributions is less important. The algorithm can thereby use fewer samples (fewer simulations of a candidate model) and still estimate a useful likelihood gradient.

## **Background**

Here we introduce important concepts in stochastic simulation algorithms, density estimation, and evolutionary algorithms.

### ***Stochastic Simulation Algorithms***

The exact stochastic simulation algorithm was first developed in (Doob 1945) and later applied to chemical kinetics in (Gillespie 1977). The makes few assumptions about the system except that the environment is well mixed.

The basic algorithm involves two steps: (1) sampling a time delay until the next reaction occurs, and (2) sampling among possible reactions which occurs. Each of these samples are dependent on the number of molecules in the current state. When there are a large number of molecules, the time until the next reaction can be extremely small. The counts of each species also influences which reaction is more likely to occur. The system is simulated by repeatedly applying reactions and incrementing time by the sampled time amount, resulting in a random walk, time-series trajectory. See (Gillespie 1977) for more details.

The exact simulation of the Gillespie algorithm becomes critically important when the number of molecules is sufficiently small. In this case, single reactions can significantly impact reaction propensities and future states (e.g. reaching a terminating state). When the number of molecules is exceedingly large, the system dynamics are approximately deterministic because a large numbers of reactions tend to average out random fluctuations.

The exactness of the Gillespie algorithm does come at a computation cost, and several methods have been proposed to improve its performance, while still preserving exactness where necessary.

For our simulations, we use the modified Poisson tau-leaping procedure that ensures that at most one critical reaction occurs per leap (Cao, Gillespie et al. 2005). The tau-leaping speeds up the stochastic simulation by estimating the number of reactions occurring during a time period tau. The value of tau is chosen such that the change in reaction propensities during tau is arbitrarily small. When the tau leap is not large enough to provide useful speed up, the algorithm defaults to an exact simulation.

### ***Kernel Density Estimation***

In order to calculate the likelihood of a the data given a candidate model, we need to estimate the probability density of the model at each data point. There are many ways to estimate probability densities.

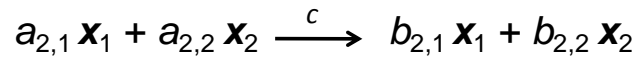
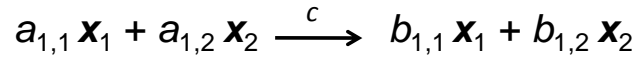
A simple method is to use a histogram. The histogram divides all samples (in our case counts of molecules after simulating a model) into a number of bins. The density is then the bin frequency divided by the bin width. Several methods exist for choosing optimal bin widths and positions (Hideaki and Shigeru 2007).

A major drawback to binned histograms however is that they are locally flat everywhere. In other words, they have no local gradient that is amenable to optimization.

An alternative to a histogram, and the method used in our experiments, is kernel density estimation (Rosenblatt 1956; Parzen 1962). Kernel density estimation is a non-parametric method to estimate probability density functions. It sums a series of kernel functions that are centered on each sample. We used a Gaussian kernel function, meaning each sample contributed a Gaussian density around its sample value. Choosing a uniform kernel for example would produce a result similar to a binned histogram.

The Gaussian kernel produces density estimates, useful for optimizing, however we still need to specify bandwidth. The bandwidth is analogous to the bin width in a binned histogram. Variable kernel bandwidth selection is the technique of selecting a different bandwidth for each sample (Terrell and Scott 1992). Variable bandwidths allow the kernels to be narrow in high density regions, capturing high details of the distribution, and wide in less certain low-density areas.

### Reactions:



... ..

### Encoding:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \quad \begin{bmatrix} c_1 \\ c_1 \\ \vdots \\ c_m \end{bmatrix} \quad \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m,1} & b_{m,2} & \cdots & b_{m,n} \end{bmatrix}$$

integers

reals

integers

**Figure 13.2.** The encoding of a solution representing a stochastic model of discrete reactions. A series of chemical reactions (top) are represented by corresponding integer coefficients and real valued rate constants for each reaction (bottom).

In our experiments, we used the square-root law (Abramson 1982) for selecting the bandwidths per sample. This technique requires an initial estimate of the density – here, we used an ordinary histogram with optimize bins chosen by minimizing the mean integrated squared error (MISE) (Hideaki and Shigeru 2007). The final result is a smooth continuous estimate of the probability density that captures both sharp and diffuse features in the distribution.

### *Evolutionary computation*

See the description in the section "Evolutionary Computation" on page 3 for more information.

### **Algorithm**

The proposed method for inferring a maximum likelihood stochastic model uses an evolutionary algorithm to search for sets of reaction channels and rates to match the

data. In this section, we describe the evolutionary encoding of candidate models in the search, and the fitness function.

### *Encoding*

The stochastic model consists of a series of reactions. Each reaction specifies an integer number for the inputs, an integer number for the outputs, and a real valued number for the reaction rate. If a reaction does not use an input, its input value is 0; likewise for outputs.

We use a fixed, maximum number of reactions for our experiments. Candidate models can opt to use fewer reactions than the maximum by setting the reaction rate to 0, or setting the inputs and outputs to 0.

Figure 13.2 summarizes our encoding for a stochastic model. It consists of a matrix of integer valued input coefficients for each reaction, a vector of real valued coefficients for each reaction, and a matrix of integer valued output coefficients for each reaction.

A random encoding is produced by filling each matrix with random integers, normally distributed with zero mean and standard deviation of 1, and filling the reaction vector with random positive real values, normally distributed with zero mean and standard deviation of 1.

The mutation operator works by randomizing each individual element with a fixed point mutation probability. The crossover operation recombines two parent encodings to form a new offspring. We use a random single point crossover on the reactions – for example, copying the first  $n$  reactions (inputs, outputs, and rate) from the first parent, and the remaining from the second parent.

The complexity of the encoding is defined as the sum of all integer valued reaction

coefficients on both inputs and outputs of the reactions.

### ***Likelihood Estimate***

Our goal is to find a maximum likelihood model. We cannot estimate the likelihood of a model explicitly, however, we can estimate the likelihood of seeing the experimental data given a specific model. This gives a measure of how well a particular model agrees with the data. In other words, we are trying to maximize the following expression:

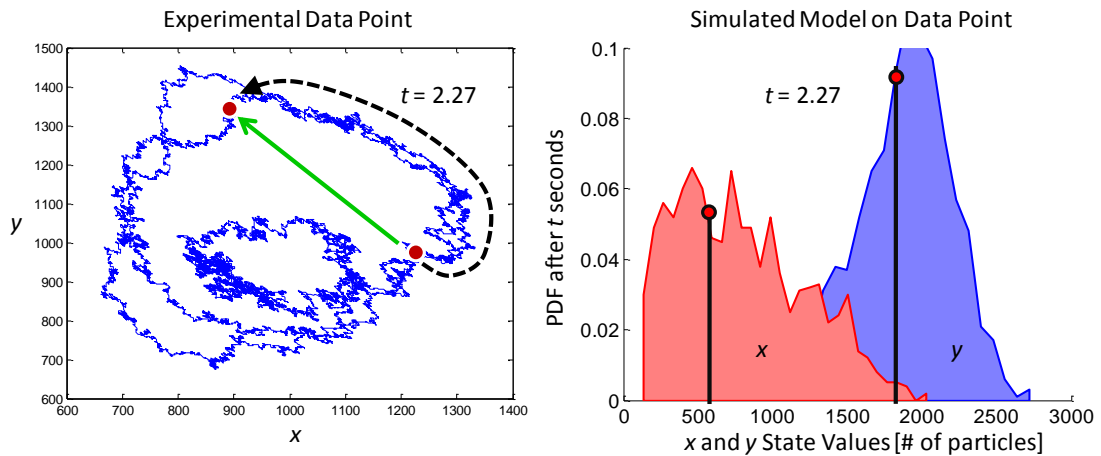
$$\text{Likelihood} = \prod_{i=1}^n P(x_i | M = m)$$

Here,  $n$  is the number of data points (measurements of a system state),  $x_i$  is a particular data point,  $m$  is a particular model, and  $P$  is the probability density of the model  $m$  at data point  $i$ . Rather than working directly with probabilities, it is numerically more stable to work with the log of probabilities, or the Log-Likelihood:

$$\text{Log Likelihood} = \sum_{i=1}^n \log(P(x_i | M = m))$$

To evaluate the likelihood, we need to estimate the value of  $P(x_i | M = m)$ . We do this by sampling the model  $m$  – that is, simulating the model over the time span from the previous data  $i - 1$  point to the current data point  $i$ .

Figure 13.3 visualizes the simulation process. The candidate model is simulated, using the previous state, until the time reaches the current state. Each simulation is then added to a kernel density estimator, described above, to estimate the probability density  $P$ . The log of the density is then summed for each state  $x$  of the system to the cumulative log-likelihood value.



**Figure 13.3. Comparing a candidate model with the experimental data. The left pane shows the hypothetical exact behavior of a system in blue, and two known measurements of the system at red dots. The candidate model is simulated multiple times, starting from the first measurement for  $t$  seconds, in order to estimate a probability distribution of the model (right). The state of the second measurement is then compared with this distribution to evaluate the quality of the model to reproduce the measurement.**

### *Fitness Function*

Ultimately we want to maximize the likelihood of a candidate model, but since we can only approximate the density function, most random models will tend to have zero likelihood and no gradient to optimize on because we cannot accurately estimate the tails of the probability density function.

Our solution to this problem is to use a two-component fitness metric. The two components are:

1. The log-likelihood as usual, and
2. The distance of the data point to the median value of the estimated distribution

When a model has near zero likelihood (e.g. lower than  $\epsilon = 10^{-6}$  in our experiments) we subtract the distance of the data point to the median value of the distribution. Otherwise, the fitness is equal to the log-likelihood. This fitness metric is

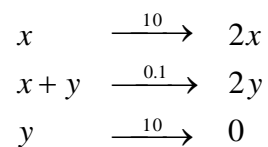


summarized in Figure 13.3.

By adding the log-likelihood component to the distance component, the fitness function remains monotonically increasing for improving models. This allows initially poor random models to move their distributions close enough to the data points such that their density estimations can be used to maximize the likelihood.

### Experiments

We perform proof of concept experiments on the basic Lotka-Volterra model (Lotka 1925; Volterra 1926). The target reactions for this system are shown below:

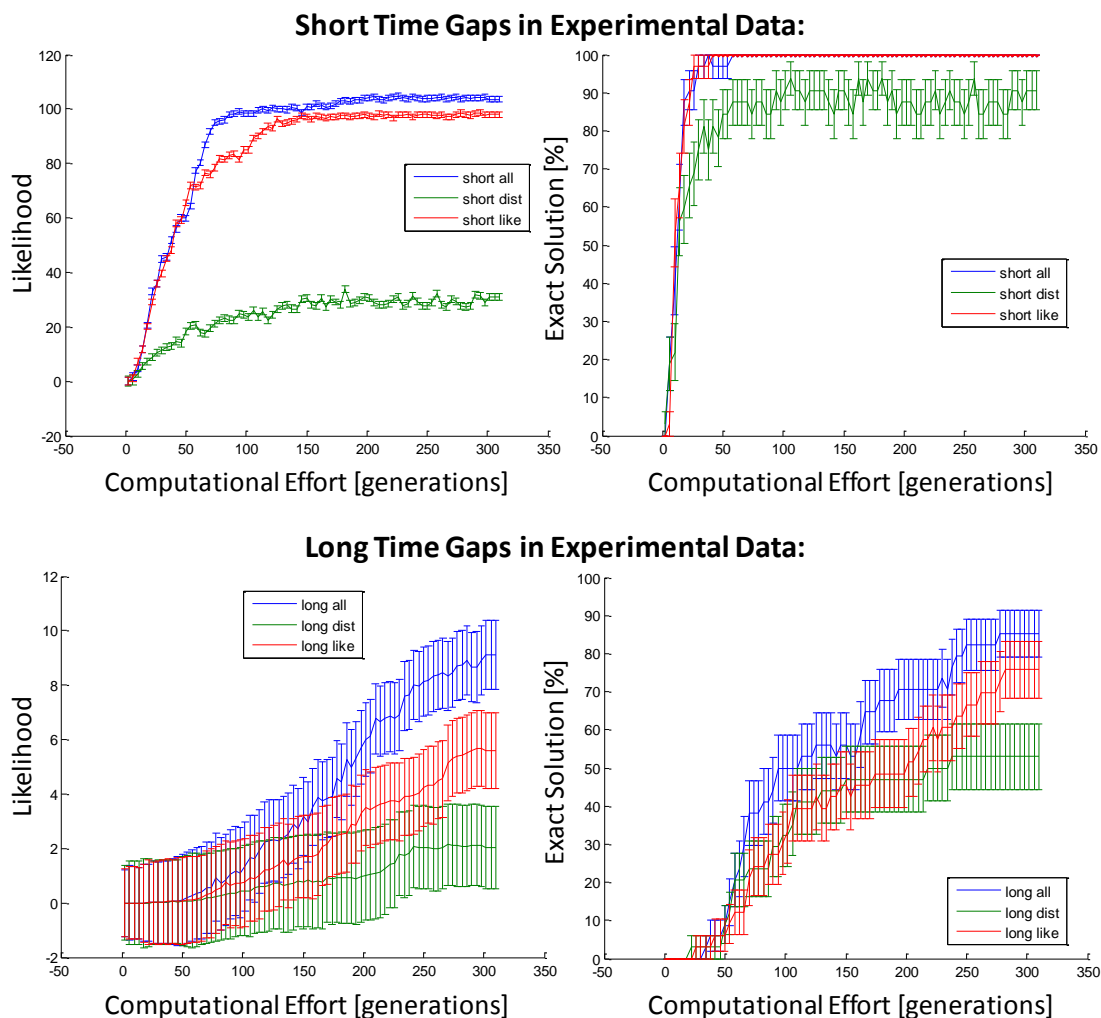


The Lotka-Volterra reactions model a predator prey system. In the first reaction, prey (represented by  $x$ ) grow exponentially. In the second reaction, prey may meet predators (represented by  $y$ ), causing a prey to die and predators to increase in number. Finally in the last reaction, a predator can die out.

We generated data sets of 10 pairs of measurements of the Lotka-Volterra system. Each pair consists of a random initial condition, followed by a measurement after simulating for a fixed time duration.

In our experiments, we compare two types of data sets, those with short time gaps, where measurements are made in short succession (time steps of 0.002), and long time gaps (time steps of 0.1) where the state of the system changes dramatically between measurements. An example of the long time gaps data set is shown in Figure 13.1(left), where each green arrow is a pair of measurements.

In the evolutionary algorithm we use a population size of 30, crossover probability of 50%, and mutation probability of 15%. We allow a maximum of 3 reactions in each model. In estimating a model density for a data point, we sample 100 independent simulations. We track various statistics of the best solution throughout each trial, including fitness on training and test data sets. We terminate all trial runs after 300



**Figure 13.4.** The search performance of the three compared fitness metrics. The top panes show performance when data points appear in rapid succession with short gaps in time. The bottom panes show performance when there are long gaps of time between data points. The left panes show the likelihood score of the best model during the search. The right panes show the percent of runs that identified the exact solution for the amount of computational effort. Error bars indicate the standard error.

iterations (generations) of the evolutionary algorithm.

We repeated the evolutionary algorithm using three different fitness metrics:

1. Log-likelihood only
2. Median distance only
3. The proposed distance and Log-likelihood metric

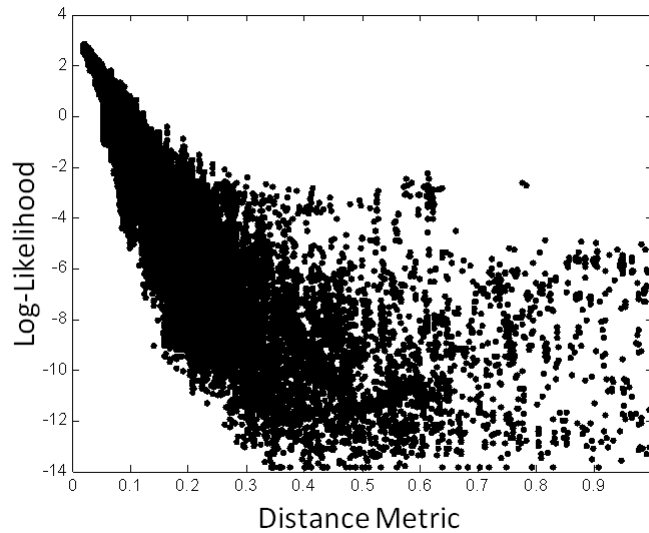
Therefore, we will be able to evaluate strengths or weaknesses of each component in the proposed metric.

## **Results**

The first results is that the evolutionary algorithm is able to find the maximum likelihood model for all three compared fitness metrics. For the short time gap data set, Figure 13.4 (top) shows that all three metrics reach approximately 90% convergence to the exact known model. Both the likelihood and hybrid metrics perform 100% convergence after 100 generations.

In terms of computation time, each generation took approximately 1 minute. Most computation cost lies in simulating various candidate models to estimate their probability densities for each data point.

On the data set with large time gaps, Figure 13.4 (bottom) shows greater differentiation between the three metrics. The two-component metric reaches the highest likelihood models and convergence, followed by the likelihood only metric. The distance metric only performs the worst.



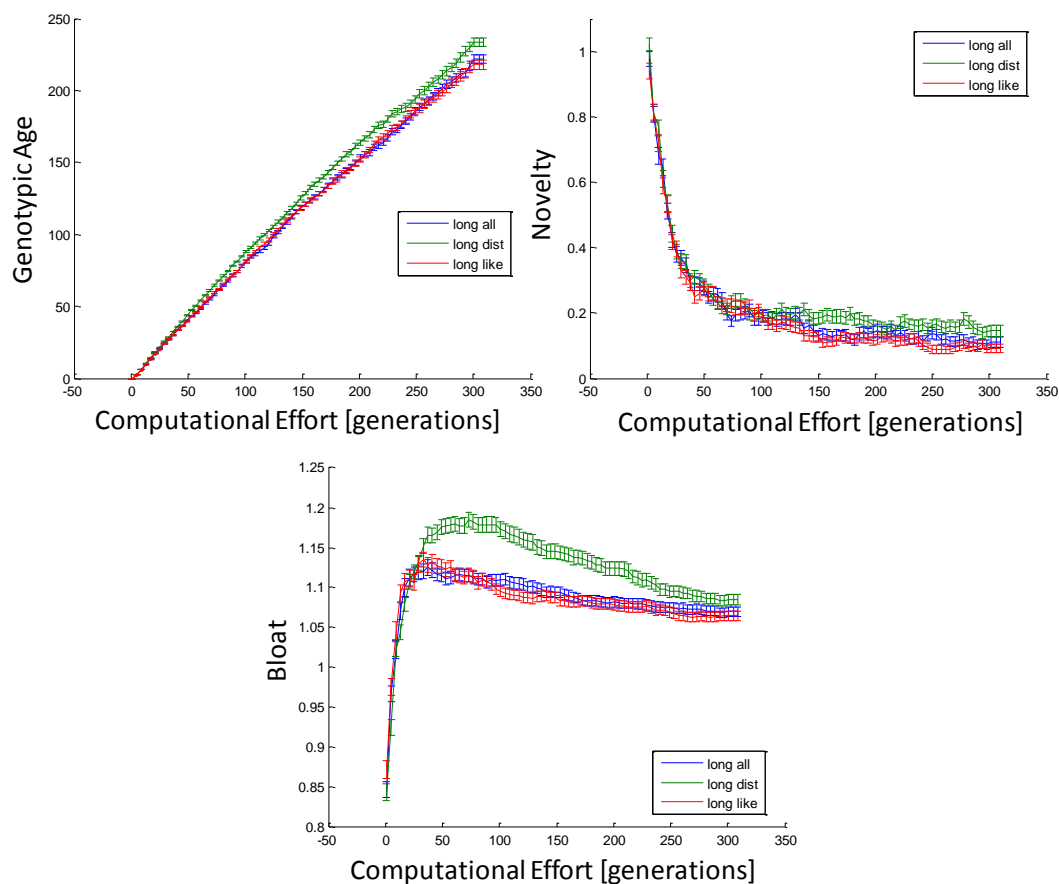
**Figure 13.5.** The relationships between the distance metric of a model and its corresponding likelihood given the experimental data. Each point in the plot is a random candidate model during the likelihood search.

Interestingly, when the time gaps are short, the performance of the two-component metric and likelihood metric only are approximately the same. This indicates that on short time gaps, the probability density of random candidate models is more likely to provided a useful search gradient, because data points are close to their initial conditions. Here, there is no benefit to using the extra distance component in the fitness metric.

However, the distance metric appears to be crucial when the data set has large time gaps (Figure 13.4). Here, the two-component metric out performs the other metrics.

Also interesting is that the distance metric alone performs very poorly. This metric allows models to get their distributions centered on the data, but does not optimize the likelihood making it inadequate on its own.

In Figure 13.5 we compare the relationship between the log-likelihood score and the distance metric. We can see that the distance is correlated with the log-likelihood, but



**Figure 13.6. Traits of the best model over time during the evolutionary search. The top left plot shows the genotypic age of the best solution (the number of generations any part of the solution existed in the population). The top right shows the novelty of the best solution (how different it is from the rest of the population). The bottom pane shows the bloat of the best solution (ratio its complexity with the target solution complexity). Error bars indicate the standard error.**

imperfect. There is large variance vertically in the log-likelihood for fixed distance, indicating that log-likelihood metric is inaccurate or at least unstable at the tails of the model probability distribution.

Finally, we collected various traits of the best solution for each algorithm during each search, shown in Figure 13.6. The first observation is that the genotypic age (Hornby 2006) of the best solution (measured in generations) is roughly equal to the total

generations on average. This indicates that the evolutionary search is not being trapped by local optima, otherwise the best solutions would appear younger as younger solutions would replace solutions in local optima. Interestingly, the distance metric algorithm tended to have the highest ages, suggesting that it avoided local optima most, perhaps by identifying an attracting region for the global optima most reliably.

The novelty of the best solution over time, shown in Figure 13.6, shows that the populations are initially very diverse before converging onto optima. But no clear difference between the compared metrics is apparent. Novelty (Lehman and Stanley 2010) is defined as the average distance summed over the reaction coefficients of a candidate solution to nearest neighbors in the current population.

In terms of bloat (Banzhaf and Langdon 2002), the algorithm starts off with a low bloat ration after random initialization. The bloat tends to increase quickly, and then fall toward a ratio of 1 (no bloat) as the best solution converges to the target (Figure 13.6). The distance only metric tended to reach higher bloat, which may be a reflection that it was less likely to converge to the target.

One final observation is that for these traits in Figure 13.6, there appears to be very little difference between the likelihood metric and the two-component metric. The key difference is only in the overall performance (Figure 13.4). This suggests that the role of the distance component is to help models move toward the data so that the likelihood component can be used, and does not impact other aspects of the population or evolutionary algorithm.

## **Conclusions**

In this chapter we introduced an automated algorithm for identifying stochastic reaction models. The proposed method used an evolutionary algorithm to identify a

maximum likelihood set of reactions and reaction coefficients. Instead of only optimizing likelihood, the proposed algorithm used a two-component fitness metric that optimized the distance of a candidate model's distribution from the data point when the likelihood was too small to provide an accurate search gradient.

The experiments indicate that the likelihood metric alone performs well on data with short time gaps in data set. However, when the data set contained large time gaps, where the state of the system evolved far from the local behavior the two-component fitness metric performed best, finding the exact target solution faster and more reliably. Observations on the age, novelty, and bloat of the best solution indicate that the algorithm avoids local optima, and could scale well with increasing complexity systems.

**Summary**

In this chapter, we analyze two general-purpose encoding types, trees and graphs systematically, focusing on trends over increasingly complex problems. Tree and graph encodings are similar in application but offer distinct advantages and disadvantages in genetic programming. We describe two implementations and discuss their evolvability. We then compare performance using symbolic regression on hundreds of random nonlinear target functions of both 1-dimensional and 8-dimensional cases. Results show the graph encoding has less bias for bloating solutions but is slower to converge and deleterious crossovers are more frequent. The graph encoding however is found to have computational benefits, suggesting it to be an advantageous trade-off between regression performance and computational effort.

**Introduction**

In this chapter, we analyze the differences between a tree and graph encoding in genetic programming. The choice of solution encoding in genetic programming can have dramatic impacts on the evolvability, convergence, and overall success of the algorithm (Franz 2006). Algorithms and encodings are often described by their bias-variance trade-off – error introduced by predisposed structure (bias), and error introduced by representative power and accommodation (variance) (David 1997; Domingos 2000; Uday and Cezary 2003). In this chapter, we examine such trade-offs more precisely, considering their representations, solution bloat, overfitting, and convergence over a range of complexity problems. In contrast with previous research, we examine these performance trends across problems with a systematically-generated range of complexities.



Tree encodings are well-known for their representative power and used heavily in genetic programming (Koza 1992). Tree encodings are generally rooted with each branch describing a unique or isolated sub-structure. In contrast, graph (or network) encodings describe groups of interacting or re-used structures.

Graph encodings allow direct re-use of subcomponents components, and can thus promote modularity and regularity in solutions. Graphs can also have a computational advantage by reducing the evaluation frequency of commonly reused structure within the solutions. However, the inherent tradeoff between modularity and regularity (Lipson 2007) suggest that reuse of modular substructures also creates internal coupling that may sometimes hinder evolvability. As a special case of graphs, tree encodings can often be adapted to graph encodings which may be more natural to the problem being solved when latent features are commonly reused.

We compare these two encoding approaches systematically using the symbolic regression problem (Koza 1992; Schmidt and Lipson 2005). Symbolic regression is a well-known genetic programming benchmark problem with precise definitions of performance and convergence. Additionally, symbolic regression provides a natural measurement of problem complexity and difficulty, allowing us to explore performance trends as problem complexity increases,

## **The Tree Encoding**

### *Structure*

The tree encoding is a popular structure in genetic programming (Koza 1992), particularly in symbolic regression. Tree encodings typically define a root node that represents the final output or prediction of a candidate solution. Each node can have one or more child nodes that are used to evaluate its value or behavior. Nodes without

children are called leaf nodes (or terminals) that evaluate immediately from an input, constant, or state modeled within the system.

Tree encodings in symbolic regression (Koza 1992; McKay, Willis et al. 1995) are termed expression trees. Nodes represent algebraic operations on children, such as *add*, *sub*, *multiply*, *divide*. Leaf nodes represent input values (e.g.  $x_I = 1$ ) or evolved constant values (e.g.  $c_I = 3.14$ ). An example expression tree is shown in Figure 14.1a.

Evaluating an expression tree is a recursive procedure. Evaluation is invoked by calling the root node, which in turn evaluates its children nodes, and so on. Recursion stops at the leaf nodes and evaluation collapses back to the root. Recursion can be computationally expensive, particularly in deep trees

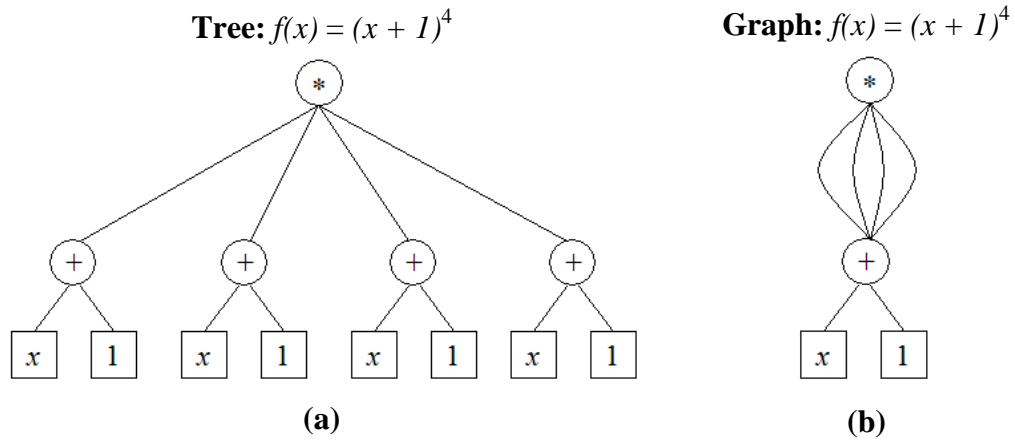
### ***Evolutionary Considerations***

Crossover of expression trees swaps two sub-trees from two parent individuals. The crossover points are typically chosen at random in each parent (McKay, Willis et al. 1995; Schmidt and Lipson 2005).

An immediate consequence of this procedure is that offspring can become extremely large by chance. For example a leaf node swapped with the root node of another parent could double the depth of the child's tree. Therefore, it is common practice to crop children or avoid crossovers that produce trees over some threshold depth.

A second consequence is repeated or duplicate structure. For example if the individual encodes the function  $f(x) = (x - 1)^4$ , the sub-expression  $(x - 1)$  must exist four times in the tree. The duplicate expressions can dominate the crossover point selection focusing recombination on  $(x - 1)$  sub-trees.

Along the same line from the previous example, duplicate expressions make mutation



**Figure 14.1. Example expressions of  $f(x) = (x + 1)^4$  in the tree encoding (a) and graph encoding (b). The graph encoding reuses redundant sub-expressions but is more susceptible to deleterious variation.**

more difficult. To produce  $f(x) = (x - 1.23)^4$  (from the previous example), the constant must be mutated 4 times.

## The Graph Encoding

### Structure

The graph encoding is similar to the tree, but child nodes are no longer unique – multiple nodes may reference the same node as its child.

Graph encodings in symbolic regression are termed expression graphs, or operation lists. Each node in the graph can represent algebraic operations, constant values, or input variables. An example graph expression is shown in Figure 14.1b.

A useful feature of graph encodings is that they lend well to efficient non-recursive representations. For experiments in this chapter, we use a list of operations that modify a set of internal variables,  $R$ . Local variable represent internal nodes in the graph and are necessary to build-up non-trivial expressions

In the list encoding, each operation in the list can reference one or more input

variables, evolved constants, or internal variables. The result from each operation is then stored in an internal variable. After all operations are completed, the final local variable is returned as output.

Avoiding recursion, without the need to cache or compile a tree expression, provides significant speed up computationally. We will analyze this improvement later in the chapter.

### ***Evolutionary Considerations***

Crossover in the graph encoding exchanges two sections of the operator list to form a child. For experiments in this chapter we use single point crossover that is chosen randomly.

The graph encoding reuses sub-expressions (multiple operations can reference the same sub-expression). Unlike the tree, crossovers in the graph are less likely to focus on redundant structure since it can be represented in a single operation (or internal variable).

For the same reason, crossover and mutation can be significantly more deleterious. An alteration to an operation producing a reused internal variable will affect all other operations which reference it. In contrast, variation in the tree encoding is localized to individual branches.

## **Experiments**

### ***Experimental Setup***

The symbolic regression algorithm and past experiments on scaling complexity can be found in (Schmidt and Lipson 2005). For experiments in this chapter, we have simply swap out the tree and graph encodings described earlier.

**Table 14.1. Summary of Experiment Setup**

Solution Population Size	64
Selection Method	Deterministic Crowding
P(mutation)	0.05
P(crossover)	0.75
Inputs	1
Operator Set	+, -, *, /, <i>sin</i> , <i>cos</i>
Terminal Set	<i>x</i> , <i>c</i> <sub>1</sub> , <i>c</i> <sub>2</sub> , <i>c</i> <sub>3</sub> , <i>c</i> <sub>4</sub>
<b>Graph Encoding</b>	
List Operations	16
Internal Variables	4
Evolved Constants	4
Crossover	variable, single point
<b>Tree Encoding</b>	
Initial Depths	1-5
Crossover	single branch swap

Parameters for all experiments are summarized in Table 14.1. Population size, mutation probability, and crossover probability are the same used in (Schmidt and Lipson 2005).

For experiments in this chapter we use correlation fitness (McKay, Willis et al. 1995) since it is a naturally normalized metric that translates well between multiple experiments and different target functions.

Evolution is stopped after the best candidate solution has converged on the training set (convergence defined later), or after a maximum of one million generations.

### ***Target Complexity***

We define complexity as the number of nodes in a binary tree needed to represent the function (Monroy, Arroyo-Figueroa et al. 2004; Schmidt and Lipson 2005). Target functions are generated randomly, and then simplified algebraically (e.g. collecting terms, canceling quotients, and evaluating constants) to give a more accurate representation of the targets minimum size.

This metric for complexity does not perfectly match problem difficulty. For example,  $f(x) = x_1 x_2 x_3$  is most likely more difficult to regress than  $f(x) = x_1 + x_2 + x_3 + x_4$  for combinatorial reasons. However, as seen in Section 7, the correlation with problem difficulty is strong and larger target functions take longer to regress symbolically on average for random functions.

### ***Random Test Problems***

A key focus of this chapter is to examine performance trends between the two encoding schemes over a range of different complexity problems. We collect results on randomly generated functions to get sufficient samples over several complexity targets.

Random targets are generated by randomizing a tree encoding. The target first simplified algebraically before measuring its complexity. Each encoding is then run on the same target functions.

The training data is generated by sampling the target function randomly over the range  $\mathbf{R}^n \in [0, 2]$  for all input variables 200 times. The test set is generated similarly by sampling over the range  $\mathbf{R}^n \in [0, 4]$ .

Results are collected over 500 randomly generated target functions, divided evenly

among complexities (1, 3, 5, ..., 19), or 50 random targets per complexity. Additionally we test on two input feature sizes: single variable and 8-variable.

### ***Convergence Testing***

Convergence is defined as having greater than 0.9999 correlation on the training set. Evolution is stopped if the best candidate solution reaches this correlation.

Note that convergence on the training set may not mean the target function has converged; the solutions may have overfit to the training data. For this reason we report convergence on the test set (test set correlation greater than 0.9999) in experimental results.

## **Results**

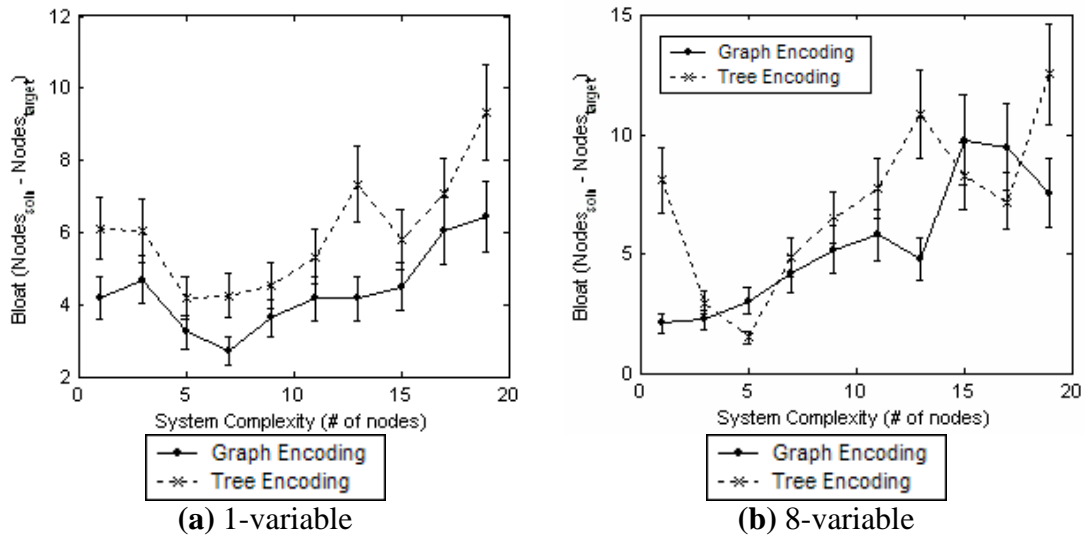
### ***Solution Complexity and Bloat***

Bloated solutions are those which are excessively complicated. In machine learning, bloat is synonymous with “overfitting” where solutions contain complex structures that do not exist in the target function to explain the fitness objective.

We measure bloat as the complexity of the regressed solution minus the complexity of the target function:

$$Bloat = (\# \text{ nodes in solution}) - (\# \text{ of nodes in target})$$

This definition of bloat will be zero if the evolved solution is the exact same size as the target (perfect case) or positive it is larger. In rare cases, converged solutions may use fewer nodes if further simplification on the target function is possible but not caught by our algebra library.



**Figure 14.2. Bloat of converged solutions for 1-variable functions (a), and 8-variable functions (b). Each point is averaged over 50 randomly generated target functions. Error bars show the standard error.**

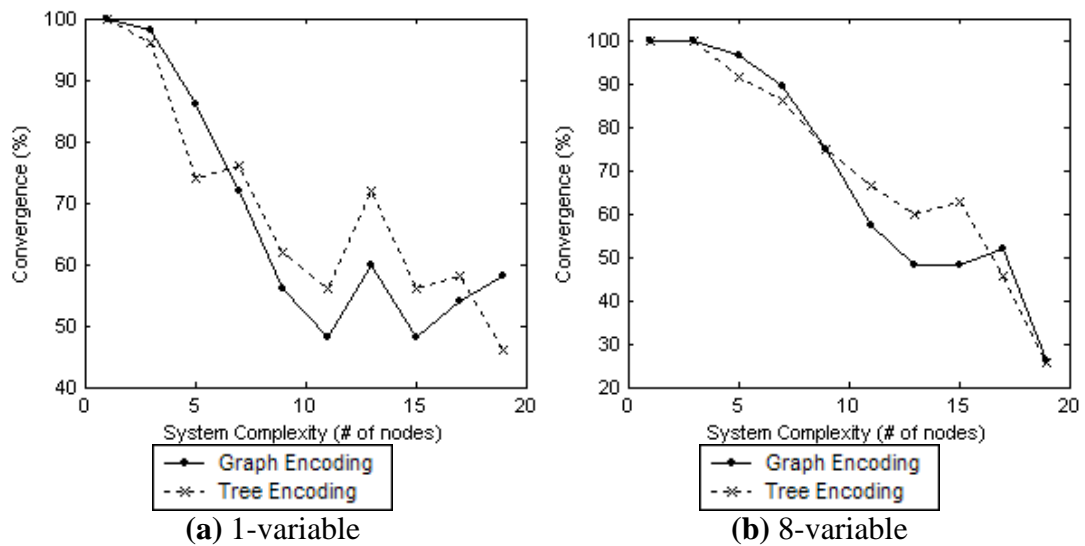
We measure the effective number of nodes in the graph encoding by converting it to a binary tree. This always increases the number of nodes but allows better comparison with the tree encoding results.

The mean bloat of each encoding type is shown in Figure 14.2 at each target function complexity. In the 1-variable case, the tree encoding has higher average bloat over all complexities. The amount of bloat (for both encodings) tends to increase with target complexity. Bloat is also higher on average in the 8-variable targets than the single variable targets.

### ***Convergence Rate***

In this experiment we measure the convergence rate for each encoding over target function complexity – the percent of runs where the best solution achieves greater than 0.9999 correlation on the withheld test set. Figure 14.3 shows the test set convergence for each complexity target function. Both encodings drop in convergence with higher complexity target functions. Each encoding is run on the same target functions.





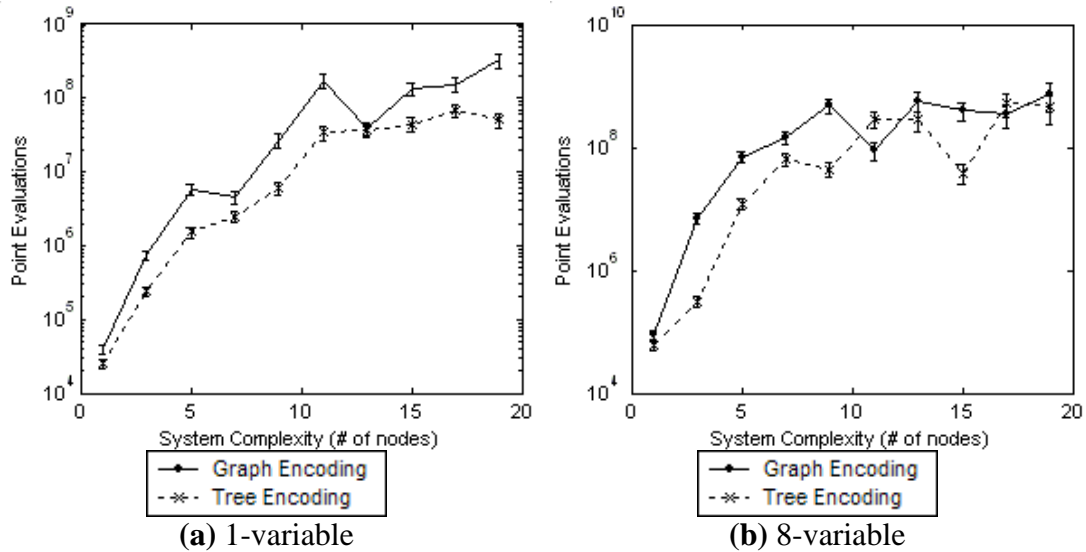
**Figure 14.3. Test set convergence versus target function complexity for 1-variable functions (a), and 8-variable functions (b). Each point is corresponds to 50 randomly generated target functions.**

The tree encoding achieves slightly higher convergence than the graph encoding over medium sized targets. However, their general trends in both the 1-variable and 8-variable cases appear to be comparable.

### *Convergence Evaluations*

In this experiment we measure the number of point evaluations before convergence on the training set. A point evaluation is a single execution of a candidate solution on a given input. Therefore, this is a metric of the total computational effort required for convergence.

Figure 14.4 shows the mean number of point evaluations to convergence for each encoding where the runs had converged on the training set. In the single variable case, the graph encoding always takes more evaluations on averaged to converge than the tree encoding. This suggests that the graph encoding is less evolvable, or perhaps more conservative considering it is less likely to bloat.



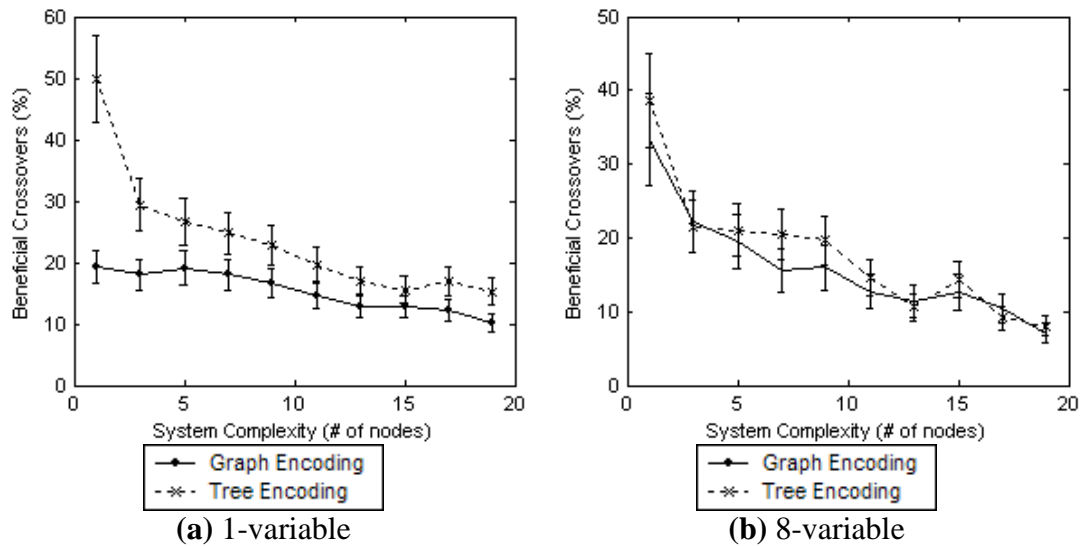
**Figure 14.4.** The number of point evaluations before convergence on the training set versus the target function complexity for 1-variable functions (a), and 8-variable functions (b). Points are averaged over 50 randomly generated target functions. Error bars show the standard error.

In the 8-variable case however, the difference in point evaluations decreases for higher complexity targets. At complexity ten and higher both encodings perform roughly the same. These figures show only runs where both encodings converged on the training set. In the 8-variable case the effort appears to require less computation, but fewer runs were able to converge before a million generations.

### *Evolvability*

In this experiment we measure the number of beneficial crossover occurring during evolution. A beneficial crossover occurs when a child achieves higher fitness than its most similar parent.

Figure 14.5 shows the rate of beneficial crossovers for both encodings over the range of complexity target functions. In the single variable case, the tree encoding experiences more beneficial crossovers than the graph encoding, particularly at low complexities.



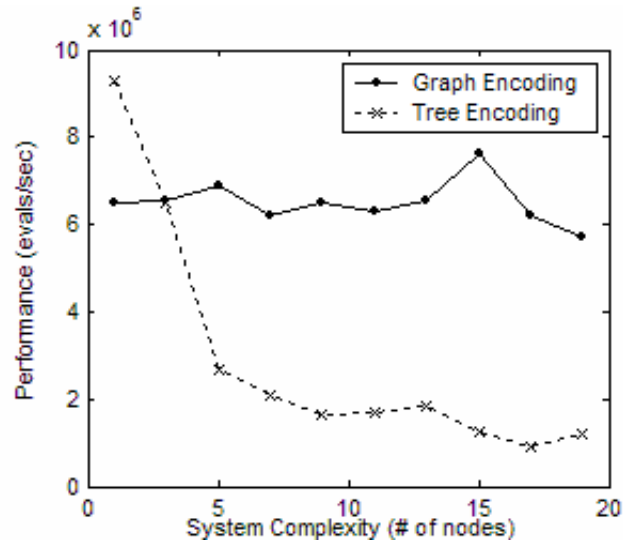
**Figure 14.5. The rate of beneficial crossovers versus target function complexity for 1-variable functions (a), and 8-variable functions (b). Results are averaged over 50 random test problems. Error bars show the standard error.**

### *Computational Performance*

In addition to evolvability, bloat, and convergence, the efficiency of encodings can have a large impact on the difficulty of problems that can be solved in practice. In this section we benchmark the tree and graph encodings.

Figure 14.6 shows the computational performance, measured in point evaluations per second over a range of complexities. The graph encoding remains roughly constant because it has a fixed encoding size. Variation still exists because it still executes operations in its list that do not affect the output.

The tree encoding is efficient on simple functions of less than five nodes. Performance drops significantly with complexity however as recursion deepens with complexity. The computational performance result indicates the tree encoding does not scale as well with complexity. At five nodes and higher, the graph encoding using an operator list more than triples the performance of the tree encoding.



**Figure 14.6. The point evaluations per second versus the function complexity.**

## Conclusions

We have compared two encoding schemes in increasingly complex problems using symbolic regression. While the tree and graph encodings are similar in application, they offer distinct advantages and disadvantages in genetic programming.

We have tested these two encodings on randomly generated nonlinear target functions, for both single variable and 8-variable input spaces.

Results show that the tree encoding solutions exhibit consistently higher bloat over all complexity targets. The tree encoding however offers slightly higher convergence rate (finding an exact fit) and time to converge, but there was no large trend difference over complexity. The tree encoding experiences more beneficial crossovers (offspring more fit than most similar parent) on single variable targets. Beneficial crossovers decrease with complexity. On 8-variable targets both encodings experienced similar trends in beneficial crossover trends. Finally, the computational comparison shows that the graph encoding is more efficient than the graph for high complexities.

From these results we conclude the graph encoding to be a attractive alternative to traditional tree based problems (e.g. symbolic regression). Graph encodings provide similar performance in convergence over a range of complex target functions and different input sizes, and do so with less bloat. The graph encoding experiences fewer beneficial crossovers and converges slightly slower, however the computational performance outweighs this drawback.

## SECTION III – INTERPRETING RESULTS

### CHAPTER 15. PARAMETER MAPPING

#### **Summary**

Recent automated scientific discovery processes hold the potential to accelerate scientific inquiry in many fields, but also present scientists with a new kind of challenge: How to assign meaning to the discovered relationships, and how to reconcile the new knowledge with current understanding. We used automated modeling to gain new insights into cellular differentiation dynamics. The process discovered a new and substantially simpler model of the dynamics of cellular differentiation of *Bacillus subtilis* that is equally predictive on unseen data. Further, it identified a new invariant, which through a process of automated-mapping was found to be closely tied to the differentiation period of the cell. This prediction was validated using a set of new experiments. We argue that beyond the value of these two specific new models to the understanding of *Bacillus subtilis*, the search for invariants and their mapping to existing knowledge may be a way of identifying governing principles of other biological systems. Just as physical conservations, such as the conservation of energy, can help understand physical processes, so can biological conservations help identify new homeostatic properties selected for by evolution.

#### **Introduction**

Increasing throughput of experimentation and data collection has placed a growing demand on automated modeling and knowledge discovery techniques (Ball 2009; Mitchell 2009; Waltz and Buchanan 2009). While recent developments in automated scientific knowledge discovery have the potential to accelerate scientific inquiry in many fields (King, Rowland et al. 2009; Schmidt and Lipson 2009), scientists will

increasingly be faced with the challenge of interpreting these models and reconciling new insights with existing knowledge.

In this chapter, we juxtapose automated modeling with the current biological understanding of cellular differentiation of *Bacillus subtilis*. We first developed a computational method for automatically generating symbolic models of single-cell dynamics using data collected from multiple cells. We then compared these data-driven models to existing, manually-derived models produced from first principles. The automatically-generated data-driven models appeared to have a markedly simpler form than the established manually-derived models, but could not be readily understood. We then developed an additional method for elucidating the meaning of the automatically-generated models by mapping components of one model to its counterpart. We begin by describing the target biological system and the computational technique, and then follow with new models generated and how these models led to new insight when compared to the manually-derived models.

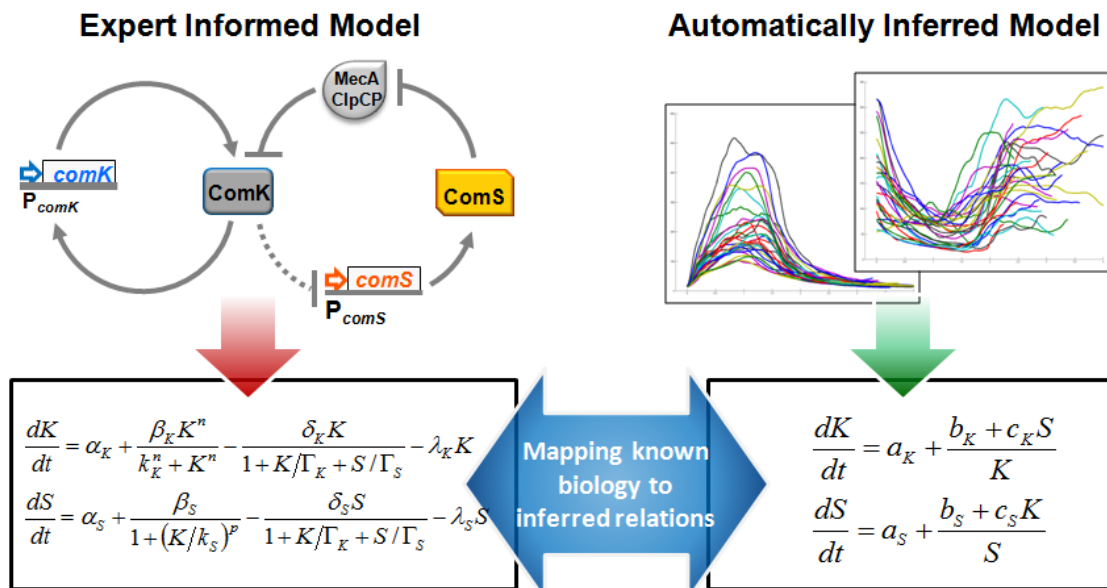
The genetic circuit that controls differentiation of *Bacillus subtilis* cells into a state of competence for uptake of extracellular DNA is well-suited for automated analysis, because it is well characterized yet poorly understood. For example, the genes and proteins comprising the competence gene regulatory circuit have been identified and characterized. Furthermore, we can quantitatively measure the dynamics of multiple components of the competence circuit simultaneously and at the single-cell level. The resulting data reveal the dynamics of interactions within the cellular differentiation circuit. Despite these features, a comprehensive understanding of how individual biochemical reactions comprising the competence circuit contribute to cellular differentiation is currently lacking. The presently accepted model for this circuit has been derived from known biochemical reactions, yielding the differential-equation

model shown in Figure 15.1(left).

New techniques for real time high resolution single-cell measurements of gene circuit dynamics can now provide new data that includes information about cell-cell variability (Figure 15.1 (right)). This presents an opportunity for automated scientific methods, which rely heavily on experimental data, to identify improved empirical models of these dynamics, and possibly new insight into the local, single-cell dynamics.

We used two types of automated modeling approaches which analyze experimental data: The first is a search for time-delay differential equation models (Bongard and Lipson 2007), and the second method is a search for invariant equations and conserved quantities (Schmidt and Lipson 2009). We then used a method we call *automated-mapping* (described below) to uncover how the automatically-generated models map onto existing manually-derived models. We perturbed the parameters of one model and generated synthetic data sets, and then fitted the automatically-generated models to those generated data sets. This process highlighted the correspondences between the parameters of the two models. Moreover, by using the perturbation itself as an experimental parameter, we could use the symbolic modeling algorithm itself to also uncover the specific nonlinear mapping between the automatically-generated models and the manually-derived models. When such a mapping exists, it shows how the manually-derived model understanding collapses to the mathematical model inferred directly from the data.





**Figure 15.1. Manually-derived versus automatically-generated biological models and the mapping challenge.** Most biological models are derived by hand using expert knowledge of the system, related systems, and qualitative understandings of the underlying biology (left). When large amounts of experimental data are available, empirical models can be inferred automatically by a computational search for the most parsimonious model that accurately predicts the dynamics (right). The automatically-generated model potentially provides new insight into the system but does not have any accompanying explanation. Our solution to this problem is to additionally learn a mapping from the known biological model to the automatically-generated model, identifying which understood parameters collapse to simpler explanations in the automatically-generated solution. Actual models and data shown.  $K$  and  $S$  represent the protein concentration levels of *ComK* and *ComS*, respectively.  $\alpha$ , and  $\beta$  terms correspond to the basal and maximum rates of protein expression, respectively.  $\lambda$  denotes the linear and  $\delta$  the enzymatic degradation rates of *ComK* and *ComS*. The meanings of the parameters on the right are unknown.

Based on the dynamical modeling and its mapping, we found that that the key dynamics of the *B. subtilis* cellular differentiation behavior can be captured in a six-parameter dynamical model, as compared to the 14-parameter state-of-the-art model. In addition, the conserved quantity search identified a previously-unknown invariant equation. We cannot tell immediately what the conserved value measures or represents. However, the mapping shows that the invariant parameters are linked to the duration of competence events in the cell, suggesting that the competence duration

may be a fixed or regulated property in each individual cell. After modifying the bacillus strain and collecting new data, we verified that the magnitude of the invariant predicts the duration of competence events observed in each cell.

Below, we introduce the automated modeling methods and the biological system in greater detail. We then analyze the resulting models and their mappings to the manually-derived model and discuss our findings and conclusions.

### **Current Biological Understanding**

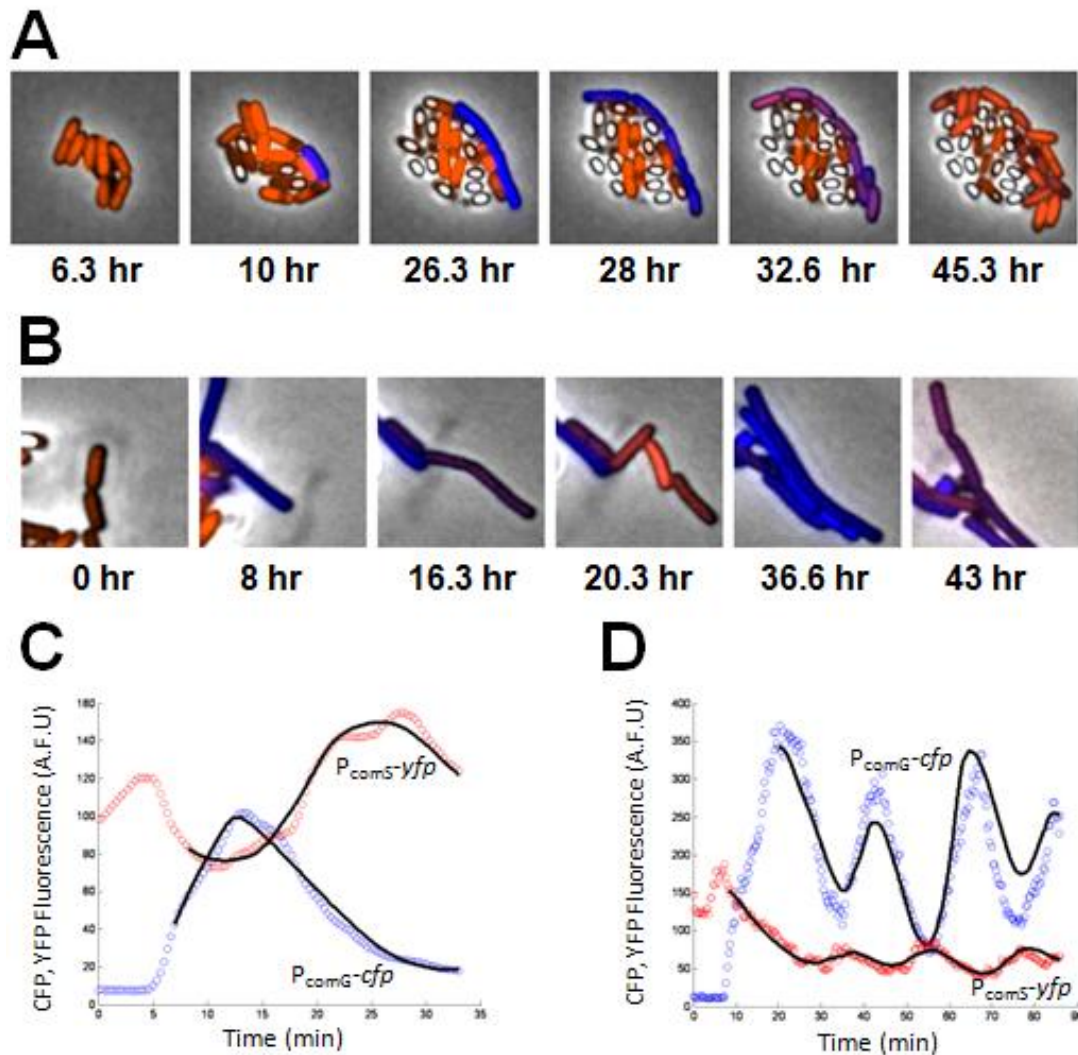
The *B. subtilis* competence system exemplifies in its simplest form the typical problems associated with developing a comprehensive and conceptual understanding of the operational principles of gene regulatory circuits. Under conditions of stress such as nutrient deprivation, *B. Subtilis* cells can transiently become competent and take up DNA from the environment and incorporate it into their chromosome. Therefore, competence is believed to provide genetic diversification and templates for gene repair.

The differentiation of cells into the competence state is triggered in an autonomous and stochastic manner. Once differentiated, cells remain in the competent state only for a transient period of time. The probabilistic initiation and transient duration of the competence state at the single-cell level is controlled by a gene regulatory circuit which constitutes a nonlinear system with excitable dynamics (Süel, Garcia-Ojalvo et al. 2006). At the heart of the competence circuit is the transcription factor *ComK*, whose expression is necessary and sufficient for competence (Sinderen, Luttinger et al. 1995; Hahn, Luttinger et al. 1996). *ComK* positively auto-regulates its own expression thereby forming a positive feedback loop (Maamar and Dubnau 2005; Smits, Eschevins et al. 2005). The cell exits from the transient state of competence via

a negative feedback loop in which *ComK* indirectly represses the expression of its activator *ComS* (Süel, Garcia-Ojalvo et al. 2006). The competing positive and negative feedback loops are described by a two dimensional model of *ComK* and *ComS*, based on the known biochemical reactions, shown in Figure 15.1(left). This model accounts for experimental observations and has been shown to be predictive (Süel, Garcia-Ojalvo et al. 2006; Suel, Kulkarni et al. 2007).

The *B. subtilis* competence behavior is well-suited for automated knowledge discovery methods because the organism is experimentally accessible. In particular, the dynamics of multiple gene circuit components can be measured simultaneously at the single-cell level using quantitative multicolor fluorescence time-lapse microscopy (Figure 15.2). However, despite these advantages, how individual biochemical reactions at the molecular level contribute to nonlinear dynamics and physiology at the cellular level remains poorly understood.

We measured the activities of *ComG* and *ComS* promoters simultaneously at the single-cell level using quantitative time-lapse microscopy, utilizing the spectrally distinct fluorescent protein reporters *cfp* and *yfp*. Transcriptional reporter constructs were integrated into standard non-essential sites of the *B. subtilis* chromosome. We followed 33 *B. subtilis* cells containing these reporters that transiently differentiated into the competence state and collected time-series trajectories of *ComS* and *ComG* promoter dynamics. Furthermore, we also utilized a genetically modified *B. subtilis* strain in which the competence circuit was perturbed to generate oscillations (Suel, Kulkarni et al. 2007). Together, the native and modified strains allowed us to record pulse and oscillatory dynamics of the competence circuit under two distinct parameter regimes, thereby providing additional information on the operation of the competence circuit.



**Figure 15.2. Transient and oscillatory dynamics of competence events in single *B. subtilis* cells.** Filmstrips in panels A and B show overlays of phase contrast and two-color fluorescence images. Blue and orange colors depict the reporter for competence  $P_{comG}$  and negative feedback loop component  $P_{comS}$ , respectively. Panel A shows a single wild type cell that differentiates into the competence state and then exits (indicated in blue). Panel B, shows cells containing a modified competence circuit (for details see text and SOM) that give rise to oscillations in competence where cells undergo consecutive events. Panels C and D depict time traces of promoter activity obtained from quantitative image analysis of fluorescent reporters during the competence events shown in panels A and B respectively. Blue and orange colors utilized in the graphics are consistent with the colors depicted in the filmstrips and time traces, where blue indicates competence and orange the activity of the negative feedback loop necessary for exit from competence.

## **Automated Modeling**

Automated modeling is a process that builds a new model of a system directly from experimental data rather than from prior knowledge or assumptions about the underlying biological mechanisms. Automated modeling can potentially provide a different or unbiased perspective on experimental observations.

The automated modeling method we used here is called symbolic regression (Koza 1992). See the description in section "Symbolic Regression" on page 4 for more information. Symbolic regression is an established algorithm that generates analytical equations for a particular experimental data set, without recourse to expert knowledge. It uses an evolutionary search (Forrest 1993) to look for the most parsimonious mathematical model (Rissanen 1978) that fits the experimental data for a given set of variables and set of functional building-blocks.

Ordinarily, symbolic regression attempts to create a single model that explains the entire data set (Duffy and Engle-Warnick 2002; Elena, Andrei et al. 2005; Bongard and Lipson 2007; Cyril and Alberto 2007). In the *B. subtilis* system however, each individual cell may behave differently due to variation in their physical size or internal composition, corresponding to parameter changes in a more general model.

In order to find a single model that captures the behavior of *all* cells in the experimental data, we created a variation of the standard symbolic regression algorithm which we refer to as *multi-set regression*. Instead of optimizing equation structures with specific parameter values, we optimize just the equation structure, while allowing the parameter values to vary for each individual cell. The figure of merit of a candidate equation model is then how well it could be made to fit the curve of each and every cell in the experimental data as illustrated in Figure 15.3 steps 1-4.

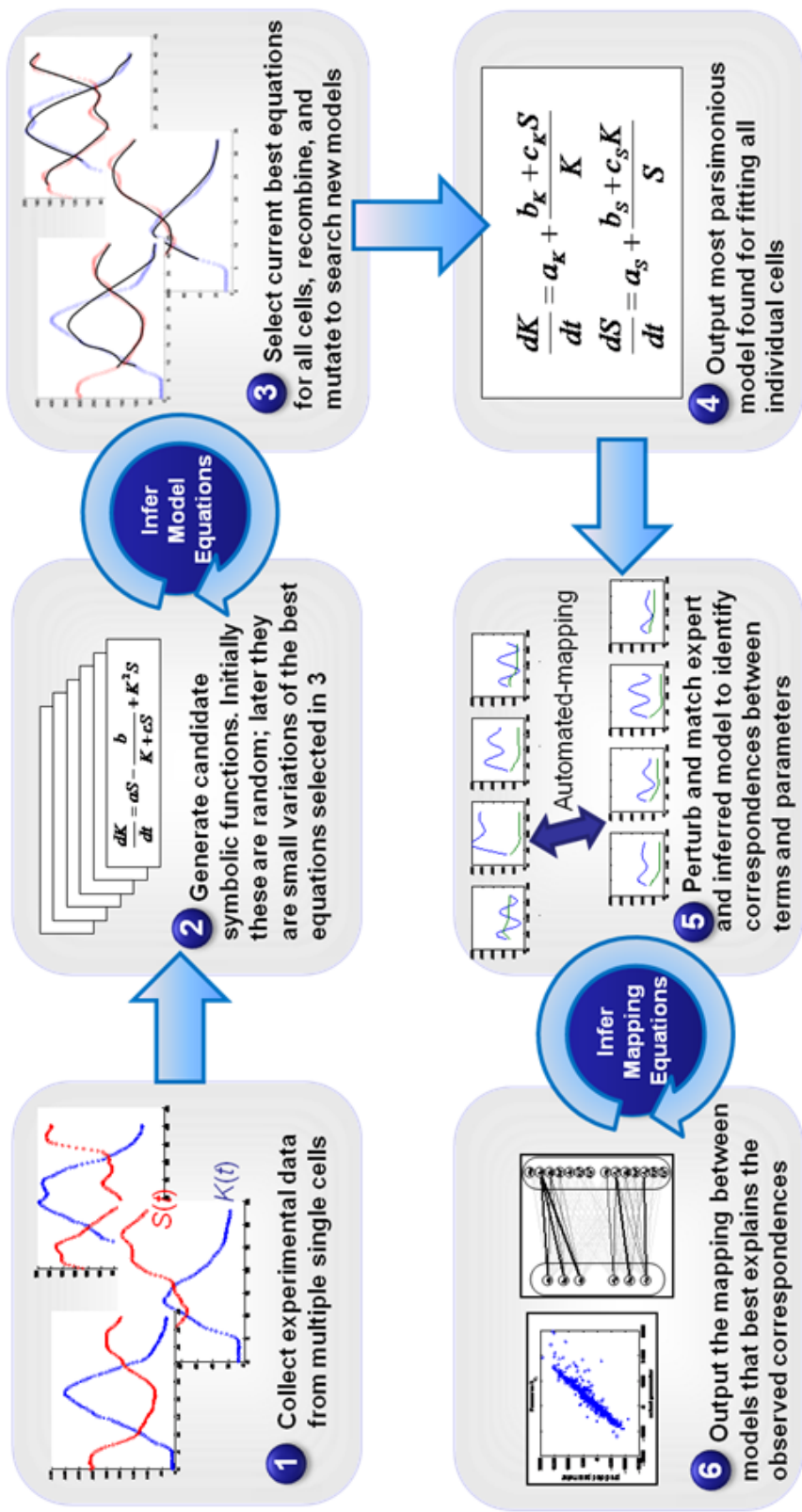


Figure 15.3. The automated modeling method attempts to model multiple cells with a single equation, and then identify a nonlinear mapping to a previous understood model. These equations contain symbolic parameters which vary for each cell, rather than constant coefficients. The algorithm searches for the most parsimonious equation which accurately predicts the dynamics observed in the experimental data using an evolutionary search. We then attempt to identify a mapping of this model to the currently understood system model by varying parameters of the manually-derived model, simulating it numerically to generate new data, and then fitting the automatically-generated model to the generated data. We then search for a nonlinear relationship between the parameters of the two models.

Additional information on the multi-set regression method is provided in the section "Multi-set Symbolic Regression" below.

### ***Dynamical Model***

Our first attempts to find a first-order dynamical model of the *B. subtilis* differentiation failed to find any accurate expressions. A lack of convergence like this typically occurs if the data is purely random, or if the algorithm does not have the correct variables or functional building-blocks (e.g. attempting to model a quotient without using division). Here, we were attempting to model the numerical derivative (Cleveland and Devlin 1988) of each variable, using only addition, subtraction, multiplication, and division.

We began finding accurate models only after allowing the search algorithm to introduce a fixed time-delay for each variable. The manually-derived biological model also required a fixed time-delay to fit all data sets.

The requirement of the time-delay in the automatically-generated model is consistent with the manually-derived model and the recent finding that *ComK* represses *ComS* expression indirectly through RapH. Such a time-delay was shown to increase the parameter regime for excitable dynamics in the manually-derived model. Therefore, the requirement of a time-delay in the automatically-generated model demonstrates that critical features of gene regulatory circuit dynamics can be identified with this approach.

The most parsimonious model found that fit the data as well as the manually-derived model using fixed time-delays is shown in Figure 15.1(right). Figure 15.2C and D show agreement of the automatically-generated model with experimental data. We further validated the generalization of this model by acquiring new data from a

genetically modified *Bacillus* strain. The initial section of these trajectories was used for the time-delay history and to optimize parameters.

Interestingly, the automatically-generated model is as accurate as the current biological model over the different dynamic regimes, but has eight fewer free parameters. The simplicity of the automatically-generated model compared to the manually-derived model suggests that several parameters involved in the production of *ComK* can be reduced to single parameters, suggesting a potentially overlooked simplicity in the generation of functional dynamics. There appears to be a small subset of parameters that account for the dynamics of the core competence circuit. Many other parameters do not seem to be as critical.

A small subset that contributes to function is reminiscent of other observations in biology such as the fact that only a few positions in proteins contribute to protein function and most others can be mutated without any measurable effect. Such properties have been suggested to be critical for evolution of biological networks.

### ***Invariant model and conserved quantities***

We also performed a separate search to detect a conserved quantity in the *B. subtilis* dynamics. Similar to the symbolic regression method, the invariant-seeking algorithm (Schmidt and Lipson; Schmidt and Lipson 2009) searches for invariant expressions that remain constant over the dataset. The motivation for this search is that in many physical systems, invariant quantities are signatures of governing laws such as the conservation of energy. The discovery of invariants in a biological system may therefore help uncover the fundamental principles governing the observed dynamics.



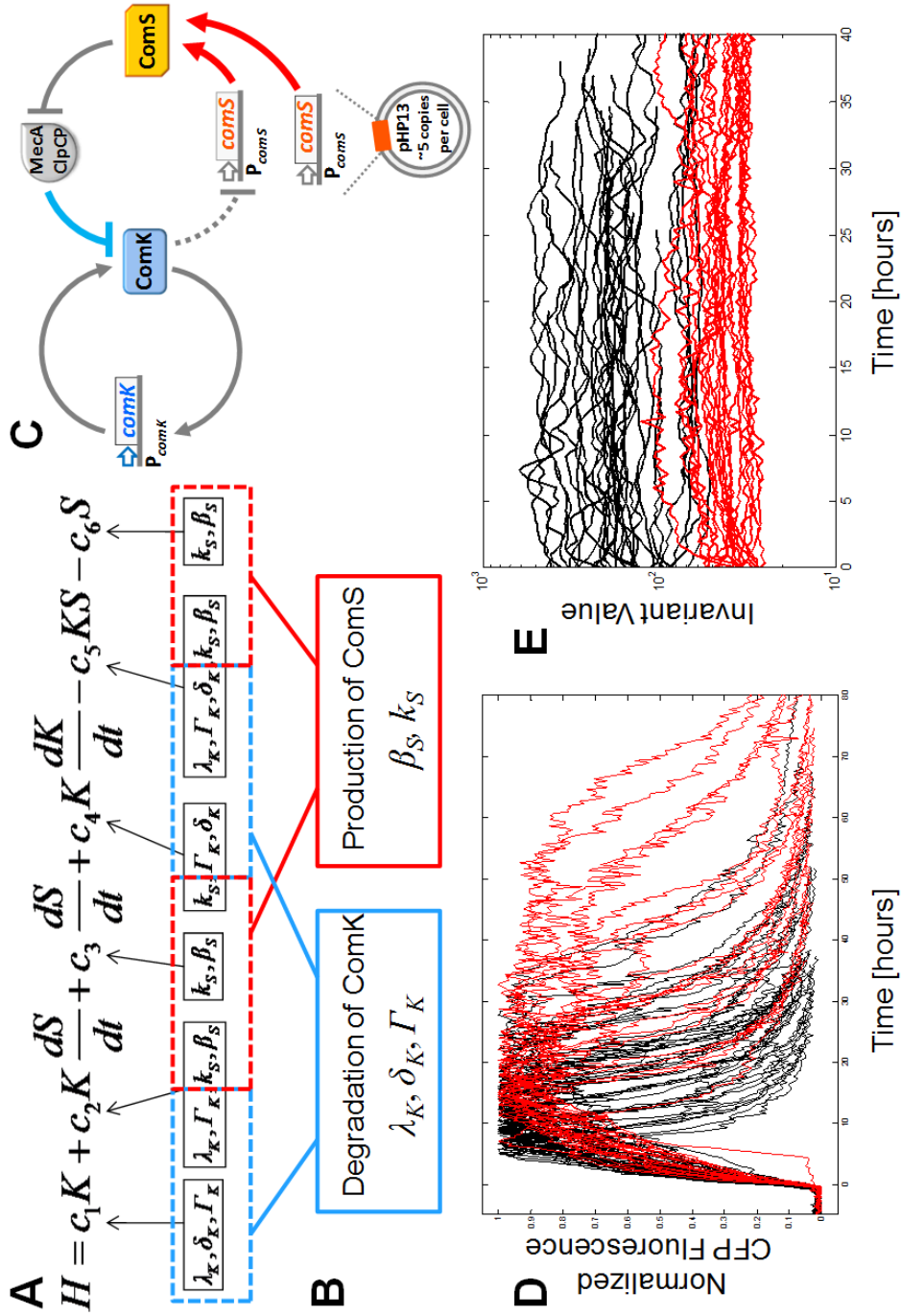


Figure 15.4. The automatically-generated conserved quantity (A) maps onto a small set of parameters in the manually-derived model (B) which correspond to the degradation of *ComK* and production of *ComS* (C). When evaluating the conserved quantity on data collected from two different types of *B. subtilis* strains (D), a sort duration strain (black) and a longer duration strain (red), the magnitude of the conserved value separates into two different groups (E), suggesting the conserved quantity is tied to the duration of competence events.

We used the invariant search algorithm to look for invariants consisting of the *ComK* and *ComS* values and their first derivatives – as might be required for some energy or momentum conservation (Schmidt and Lipson 2009). Among the candidate conservations, the function  $H$  shown in Figure 15.4A was the simplest relation and also remained invariant even on the forward experimental data of the mutated strain. Figure 15.4 also shows the invariant  $H$  plotted for several cells (pane D) of two different *B. subtilis* strains which are plotted in pane E. There is some variance in the conserved value for each cell which scales with the magnitude of the conserved value.

We fit this invariant to all data sets for both the wild and mutated strains. Since we do not know what the units and offset of the invariant are, we normalized each fit by arbitrarily fixing the last coefficient,  $c_6$ , to one. While there exists some residual variance - either from noise or approximations in the conserved value - the normalized conserved values show a clear separation between the wild and mutated strains, with very little overlap (Figure 15.4E). In fact, given data from an unknown strain, the magnitude of the conserved quantity could be used to predict which strain the cell belongs to.

### **Mapping to Current Biological Understanding**

The automated modeling results gave two previously-unknown descriptions of the experimental data: a substantially simpler dynamical model, and an unknown conserved quantity. The difficulty is how to explain and interpret these models in order to gain new biological insight. In essence, we have new answers derived from experimental data, but without any accompanying explanations.

Our solution to the interpretation challenge is to learn a *mapping* – from the current manually-derived biological model, to the automatically-generated data-driven model.

The mapping we are interested in is the relationships between the free-parameters of the manually-derived model and the free-parameters of the automatically-generated models. If a simple mapping exists, it can show how parameters in the manually-derived model collapse to the simpler automatically-generated dynamical model, and which qualities of the known biology affect the automatically-generated conserved quantity.

***Automated-mapping - using model perturbations as "experiments"***

We refer to the method for learning the parameter mapping as *automated-mapping* between two models. The basic process, summarized in Figure 15.3 steps 5 and 6, starts by simulating the manually-derived model numerically with random parameter variations. The automatically-generated model is then fit to each simulated trajectory. The result is a set of parameter values for the manually-derived model, and the corresponding parameter values for the fitted automatically-generated model. We repeat this process for several hundred random parameter variations, thereby generating a dataset of matching parameters of both models.

We first looked at linear correlations between the manually-derived dynamical model parameters and the automatically-generated model parameters. Figure 15.5B shows the strength of the correlations in a bipartite graph. The correlations suggest that each parameter in the automatically-generated dynamical model co-varies with a small number of parameters in manually-derived model. Interestingly, some parameters of the manually-derived model appear to have little influence on automatically-generated model and its dynamics, and therefore are apparently irrelevant to explaining the observed behavior in this regime.

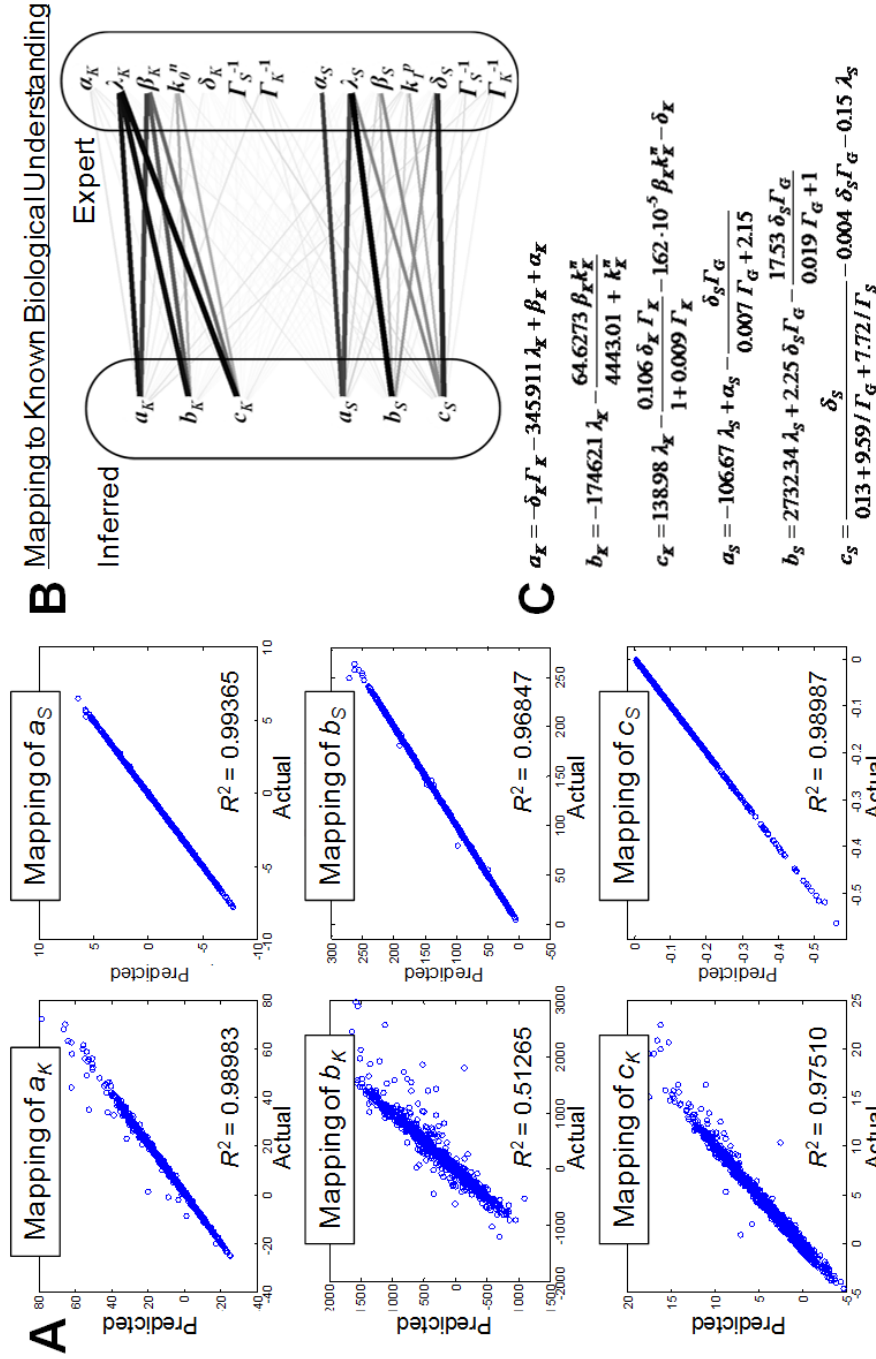


Figure 15.5. The mapping between the manually-derived model and the automatically-generated dynamical model connects the simpler data-driven model with the current biological understanding. The bipartite graph (B) shows the linear correlation strengths between model parameters – automatically-generated model parameters are on the left side, manually-derived model parameters are on the right side. The nonlinear mapping (C) shows that multiple parameters of the manually-derived model collapse to those in the simpler automatically-generated model. The parameter plots (A) show that the mapping is in strong agreement with the automatically-generated model over a wide range of parameter values.

The linear correlation shown in Figure 15.5B are averaged across a large area of the dataset, but the strengths of the correlations vary depending on the specific regime of the data. These fluctuations suggest that the relationships between the parameters are nonlinear. To investigate this further, we used the same automated model search algorithm to find relationship between the parameters of the two models, essentially using the parameter variations as "experiments".

The nonlinear mapping (Figure 15.5C) showed high accuracy; predicting the automatically-generated model parameters from the larger manually-derived model parameters with goodness-of-fit of over 0.95 for most parameters (Figure 15.5A). This suggests that the *B. subtilis* cellular differentiation dynamics are, in fact, operating on a simpler manifold with reduced dimensionality.

For *ComK*, the automatically-generated model correlates linearly with the parameters of the manually-derived model that describe the maximum production and linear degradation of *ComK*. However, for *ComS*, parameters of the automatically-generated model exhibit less correlation with production terms of the manually-derived model, and much more correlation with the degradation of *ComS*. Therefore, the production of *ComK* and the degradation of *ComS* appear to account for most of the nonlinear dynamics of the competence circuit. Only a small subset of parameters accounts for the data, which is similar to observations made in proteins and metabolic networks. This suggests perhaps a common evolutionary solution to selection pressures.

This key insight from the mapping indicates which parameters of *ComK* and *ComS* contribute most to the dynamics of the competence circuit. These results also suggest that perturbations of those parameters should give greatest effects.

## Conserved Quantity Mapping

The mapping found for the unknown conserved quantity (Figure 15.4A) using the automated-mapping procedure also provides insight into the meaning of the conserved quantity. Similar to the dynamical model mapping, we fit the invariant to the data generated from the manually-derived model, using symbolic regression to identify the nonlinear relationship between the invariant parameters and the manually-derived model parameters.

The mapping shows that the conserved quantity only depends on two types of understood parameters of the system: parameters controlling the degradation of *ComK*, and parameters controlling the production of *ComS* (see Figure 15.4B and C). In fact, these parameters are known to impact the duration of competence events in the *B. subtilis* system. The duration of transient competence events are determined by the *ComS* mediated negative feedback loop. The longer it takes for *ComS* concentrations to decrease, the longer the duration of competence. Parameters describing the production rate and concentration of *ComS* can therefore affect the duration of competence events (Suel, Kulkarni et al. 2007). Therefore, the mapping suggests that the conserved quantity is related to competence durations.

We tested this prediction by looking at the invariant evaluated on data collected from the wild type and a modified strain with higher expression of *ComS* (Figure 15.4D). Increased production of *ComS* in the modified strain was accomplished by introducing a copy of the native *ComS* promoter driving *ComS* into a plasmid maintained at five copies per cell. Effectively, this modification resulted in a six fold higher production rate of *ComS* ( $\beta_S$ ) compared to wild type. The invariant values obtained from the competence dynamics recorded from the wild type and modified strains cluster into two groups. High magnitudes for the short duration wild type, and low magnitudes for

the long duration mutated strain. In fact, the separation is clear enough that the invariant magnitude could be used to predict which strain an unknown cell belongs to and therefore its expected competence duration. These results confirmed our hypothesis based on the mapping that the conserved quantitative is related to competence durations.

The key insight from the conserved value and its mapping is that competence duration is tied to a conservation taking place in each cell. It has recently been shown that *B. subtilis* competence durations determine physiological function (Ça atay, Turcotte et al. 2009). Specifically, the duration of competence has been demonstrated to dictate the efficiency and range of DNA concentrations over which the competence circuit can perform its biological function. It is thus noteworthy that the conserved property identified here maps to parameters governing this critical property of competence.

## **Conclusions**

In this chapter, we have identified a simpler model of the dynamics of cellular differentiation of *Bacillus subtilis*, that is equally predictive on unseen data. This result demonstrates a useful application for reducing the complexity of mathematical models describing biochemical interactions. We further proposed the search of invariants as a way to uncover the natural laws governing the dynamics of this system. Indeed an invariant was discovered and was found to be closely related to the differentiation period of the cell. This prediction was validated using a new set of experiments. The search for invariants may be a way of identifying key principles of other biological systems as well. We suggest that the ability to identify such conservations can be informative for understanding increasingly complex systems in the future.

A fundamental question is whether algorithmic methods for modeling and

hypothesizing about experimental systems can ultimately be human-competitive: Can such methods produce elegant and predictive models on par with human experts, and if so, will and how could human experts understand these models. In this chapter, we have shown one of the first instances of an algorithm producing a concise, human-readable model that is consistent with a large amount of experimental data, and is substantially simpler than a recently published model for the same phenomenon. But that accomplishment only led to a new challenge: How to assign meaning to the resulting models and reconcile them with existing knowledge. Our solution was to use the automated-modeling process itself to find relationships between the new model and existing knowledge, by using model perturbations as "experiments". We believe that this kind of hurdle will become increasingly challenging as the use of automated modeling algorithms becomes more prevalent. The need for new methods to help machines "teach" their findings to humans, for example by drawing analogies to known information, may be essential to long term progress in science, and become a new frontier for Artificial Intelligence research.

## **Methods**

### ***Multi-set Symbolic Regression***

Models are encoded as an equation and a set of parameters for each unique set of data points (measurements of a single bacterial cell) in the data. Our automated modeling method is based on the symbolic regression algorithm (Koza 1992). See the description of section "Symbolic Regression" on page 4 for more detail on this technique.

Symbolic regression has been used to model explicit (Duffy and Engle-Warnick 2002; Elena, Andrei et al. 2005; Cyril and Alberto 2007) and dynamical systems (Bongard



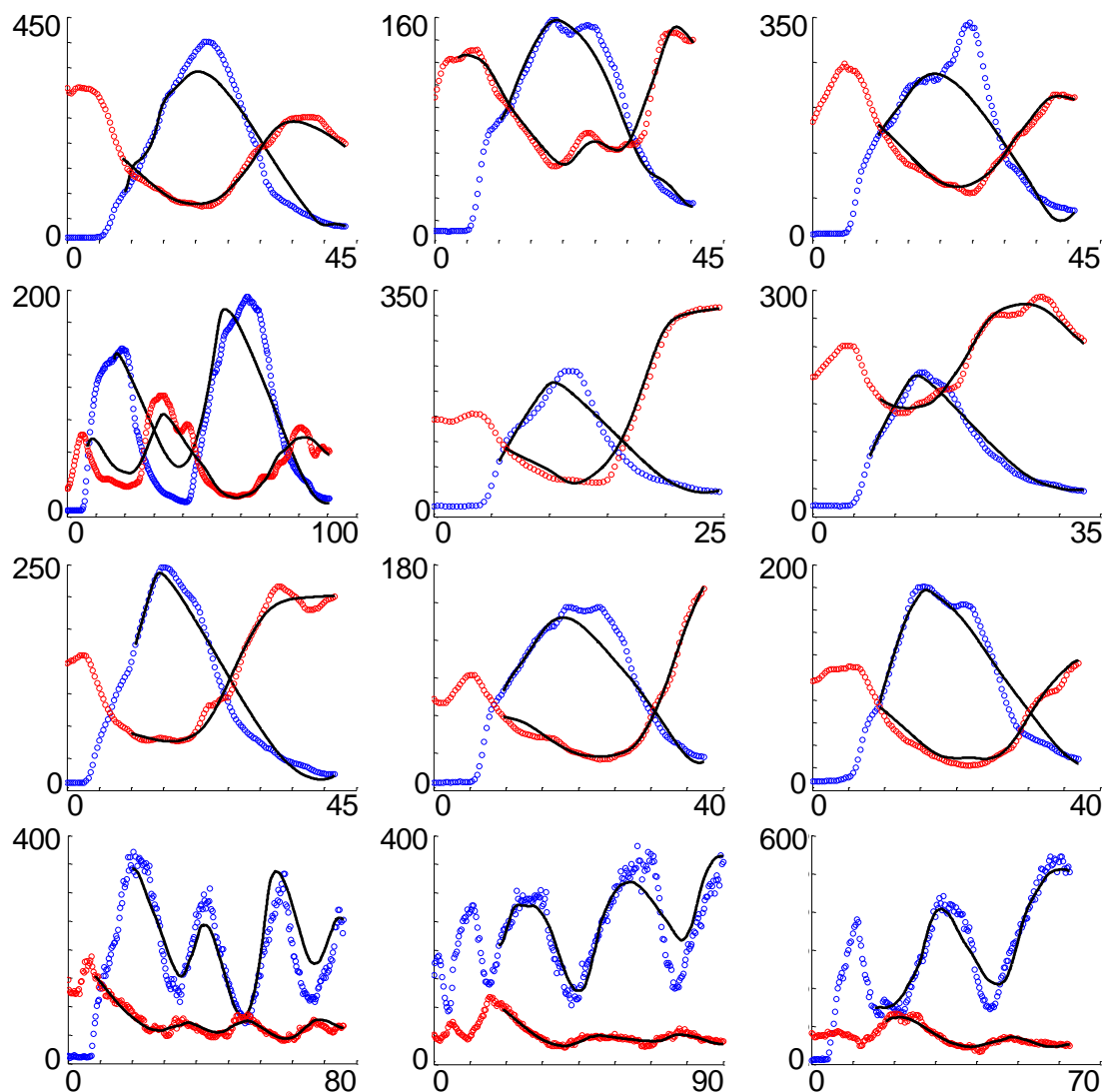
and Lipson 2007) in past research, it does not ordinarily take advantage of multiple dataset from unmatched sources, such as data recorded from multiple cells, each cell with different parameters such as physical size. In order to use multiple datasets at once to get a large enough description of the system, we developed a multi-set regression method. The method searches for a single equation set that can be fitted well to each data source independently (e.g. each individual cell), requiring only parameter adjustments, but no change in form.

### ***Model Selection***

We selected the automatically-generated model by considering the *Pareto front* (Kung, Luccio et al. 1975; Parke, Ryan et al. 2007) produced by symbolic regression between model complexity and its accuracy on the experimental data. Complexity is measured as the inverse of number of terms in the expression. Equations that are both simple and accuracy are the most challenging to find and identify, and their behavior is more interesting (Schmidt and Lipson 2009). In particular, the most interesting solutions on this frontier appear at cliff points, where the predictive ability to increases and then plateaus (Edwin and Jordan 2003; Gregory, Denis et al. 2003).

### ***The Inferred Dynamical Model***

We performed the multi-set regression technique using data collected from several different cells. The top rows of Figure 15.6 show data from different cells used to search for the model. The fit of the automatically-generated model is shown in solid black lines. The automatically-generated model fits each cell, capturing their key dynamics, despite the inherent stochastic behavior of the system.



**Figure 15.6.** Collected data and the fit of the automatically-generated dynamical model. *ComK* fluorescence (AFU) is shown in blue dots, *ComS* fluorescence (AFU) is shown in red dots, and the automatically-generated model is shown in black for each. The automatically-generated model was found using data from the top four rows. The bottom row shows that the model generalized to other behaviors such as oscillating competence events.

Fitting these data sets with a first-order model required a time-delay in the dynamical model, as described in the main text. It may also be possible to model this data using a second-order (or higher-order) model however, we were unable to find any simple second-order models that generalized to other data sets. Calculating multiple derivatives from the data set is difficult numerically, especially when estimating initial

conditions. Also, second order systems are less common in the chemical and biological context.

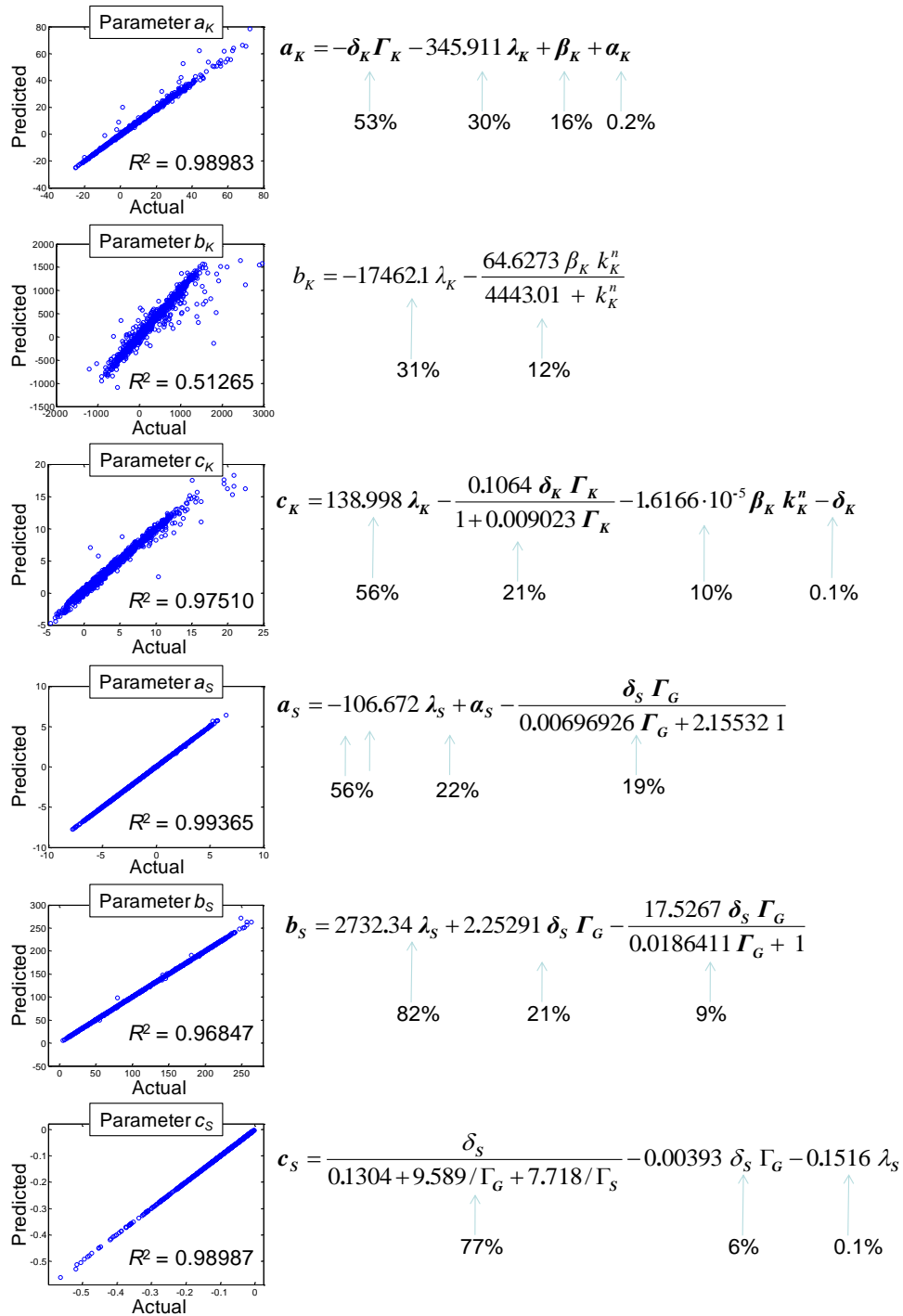
The model is fit to this data by sweeping the time-delays and least-squares fitting the model to the numerical derivative of the data for each variable, *ComK* and *ComS*. We then integrated the model using the DDE23 time-delay differential equation solver – specifying the absolute error tolerance and relative error tolerance to  $10^{-9}$ .

Figure 15.6 also shows data not used to find the model (bottom row). Here, the model generalized to different behavior from the training data to model oscillating competence events.

### ***Nonlinear Dynamical Model Mapping***

We used the automated-mapping method to find an equation relating each parameter of the automatically-generated dynamical model to the parameters of the manually-derived model. We generated data for each parameter by simulating the manually-derived model with randomly perturbed parameter values and fitting the automatically-generated model to each simulated trajectory. We then searched for an equation to predict the value of the automatically-generated model parameters based on those in the generating manually-derived model.

The resulting mapping for the dynamical model is shown in Figure 15.7. The search identified a simple mapping equation for each parameter with high goodness of fits. Based only on the parameter values of the manually-derived model, the mapping can predict the optimal fitted parameter values in the automatically-generated model with  $R^2$  values over 0.95, with the exception of parameter  $b_K$  which was 0.51.



**Figure 15.7.** The parameter mapping relating the parameters of the expert biological model and the automatically identified dynamical model. The left plots show the predicted parameter value in the automatically-generated model based on the parameters of the expert model versus the actual best fit parameter of the automatically-generated model. The parameter equations found are shown to the right. The percent shown for each term indicates the percent of the variance explained by each term.

It is interesting that such an accurate mapping exists. The two models could just as easily fit the same data in discontinuous or random ways. Instead, the mapping suggests equivalence between the two, described by the mapping equations in Figure 15.7.

### ***Conserved Quantity Mapping***

We also used the automated-mapping method to identify a nonlinear mapping between the automatically-generated conserved quantity and the manually-derived biological model. We simulated the manually-derived model with varying parameters to collect synthetic data, then fitted the invariant to each simulated trajectory. We then looked for an equation modeling the resulting fitted parameters in the conserved quantity as a function of the parameters in the manually-derived model.

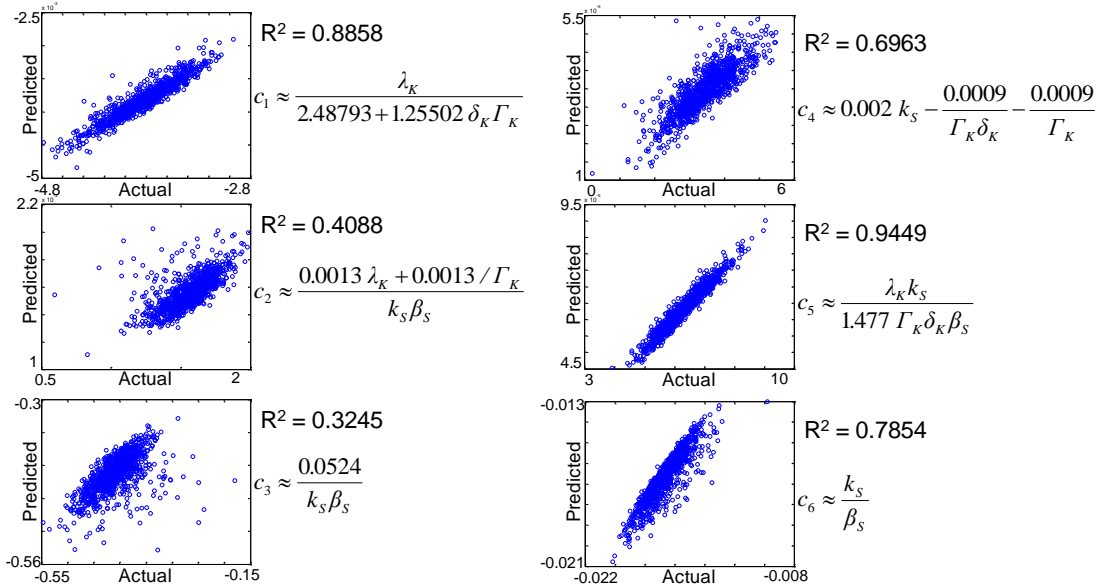
Figure 15.8 shows the resulting map for the automatically-generated conserved quantity. The conserved quantity parameters were more difficult to model than the dynamical model. This is likely a result of higher sensitivity and variance when fitting an invariant equation. The mapping however still shows strong correlations.

The result of the conserved value mapping is that we now have a method to directly calculate the conserved value from the manually-derived model directly without the need to tune parameters – they are explicitly prescribed by the mapping.

### ***Interpreting a Conserved Quantity***

Many conserved quantities correspond to a fundamental physical or natural law – such as conservation of energy or momentum. However, we are not certain what the automatically-generated conserved quantity represents in the competence circuit.

In developing our analysis of the unknown conserved quantity, we make many



**Figure 15.8.** The parameter mapping relating the parameters of the expert biological model to the automatically-inferred conserved quantity. The left plots show the predicted parameter value in the conserved quantity of the mapping versus the actual best fit parameter of the conserved value. The parameter equations found are shown to the right.

comparisons with a known conserved quantity, such as conservation of energy in a pendulum. We collected data both from a real and a simulated double pendulum (Schmidt and Lipson 2009) and apply the same types of analysis to the total energy equation of the double pendulum. This allowed us to compare the unknown conserved quantity with an understood conservation both with and without noise or loss.

Conserved quantities are often difficult or unintuitive to understand. In fact, many conserved quantities cannot be directly observed. For example, the concept of energy is abstract. In the double pendulum, we can tell that the conserved quantity (total energy) is predictive of magnitudes of the velocities of the pendulum and the maximum heights it reaches. But we cannot directly measure it; it has to be inferred from other measurements. It could have an arbitrary offset, and possibly, arbitrary scale; yet still be predictive of the dynamics of the double pendulum.

Similarly, the conserved quantity automatically-inferred for the *B. subtilis* competence system is predictive of the duration of competence events in each cell. It is quite possible that this quantity is also abstract as in the double pendulum and we may not be able to interpret this quantity any better than we can interpret and understand the concept of energy.

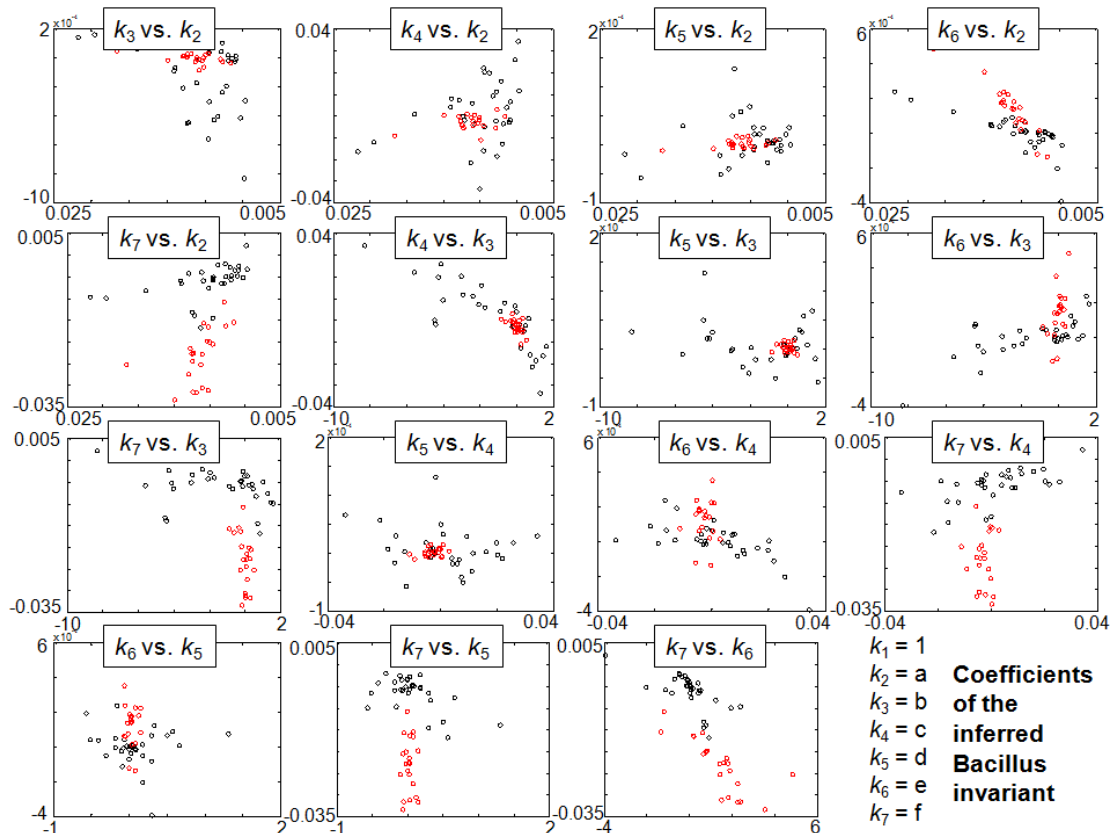
Nevertheless, we know that the conserved quantity is tied to the competence duration, and that the duration can greatly impact fitness and adaptability of the cell. It is likely that optimal durations are controlled in the cell or at least selected for by evolution. Therefore, we could interpret the conserved quantity as a control value of each cell for the competence durations. However, the scale, offset, or units we define for this value could be arbitrary; as with energy.

### *Normalizing Unknown Conserved Quantities*

One challenge when analyzing and comparing unknown conserved values is that they are invariant to scaling and offset. For example, if the formula  $f$  is conserved, so is the formula  $af + b$  where  $a$  and  $b$  are any real constants. The key problem is that we do not know the “units” of the conserved value. Therefore, we need a method for normalizing each fitted conserved quantity – removing the scale  $a$  in the previous example.

One way to normalize the scale is to divide the entire invariant equation by one of the coefficients that appear linearly in the formula since these will also contain the scaling factor. Ideally, we could divide by the scale exactly, but the coefficients also contain the parameter of that coefficient. Normalizing by different coefficients produces different scales and different orderings depending on the parameter used.

One way to visualize this problem is to plot the coefficient values of the conserved



**Figure 15.9. The clusters of coefficient values of the unknown conserved quantity equation colored by the *B. subtilis* strain. Each plot shows a projection onto a different pair of coefficients.**

value formula for both the wild and mutated *B. subtilis* strains. In Figure 15.9, we plotted pairs of coefficients (or 2D projections) of conserved value formula fitted to the experimental data collected from both the wild and mutated strains.

In these projections we can see that the wild and mutated types form distinct clusters based on the coefficients of the invariant. In several of the projections we can even separate them by a 2D hyper-plane of coefficients.

In particular, all projections shown in Figure 15.9 that have coefficient  $k_7$  as an axis appear well separated. This suggests that  $k_7$  alone is useful for normalizing with, though combinations may be even better. In the main text, we show the conserved



value after normalizing by  $k_7$ .

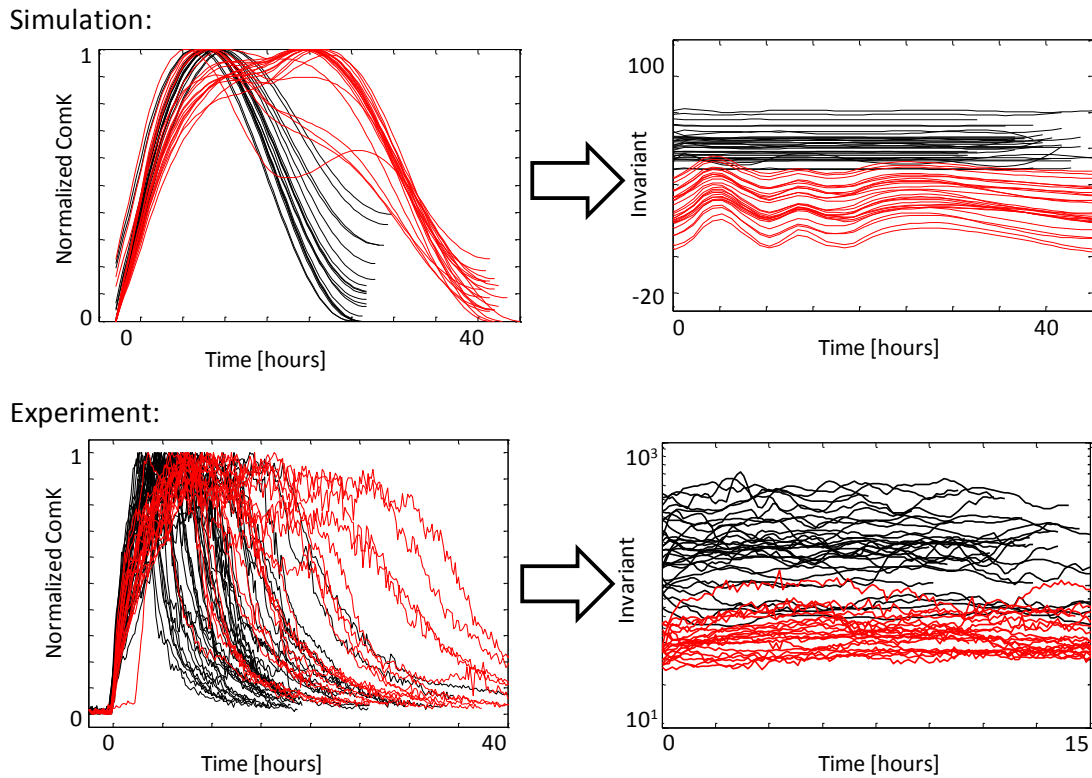
### ***Sampling Parameters Values for Automated Mapping***

In the automated-mapping technique, we use the bacillus models to generate data by simulating the system with different model parameters. This allowed us to compare the parameters of each model over many different data sets. Here we detail the procedure used to generate data on the parameters.

We started by fitting the manually-derived model to one of the experimentally collected cell data sets. For this comparison we fitted to the oscillatory data set shown in the bottom left of Figure 15.6 which happens to have more interesting dynamics over a longer period of time. We fit the manually-derived model by sweeping the time-delays for each variable, and using nonlinear regression to fit the numerical derivatives of the data for each variable. We use the beginning of the experimentally collected data as the initial time-delay history, interpolating between data points as necessary.

Next, we estimated the valid ranges of the parameters in the manually-derived model. We did this by sweeping the value of each parameter individually, holding the other parameters to their fitted values, until the system became unstable or exceeded experimentally observed ranges in either variable. This range also indicates the relative impacts of each parameter that allows us to perturb all parameter equally.

We simulate the manually-derived model multiples times varying the parameters in Matlab using the DDE solver with absolute error and relative error tolerances set to  $10^{-9}$ . For each sample, each parameter is modified by a random percent between zero and 25% of the parameter's valid range. We collected a thousand 30-hour trajectories.



**Figure 15.10. Verifying the perturbations of the models with the physical changes in the wild (black) and mutated (red) strains. Pertubing only the parameters that correspond to production of *ComS* in the simulated model produces similar changes to those seen in experiment.**

Finally, we take the automatically-generated dynamical model and fit it to each of these simulated trajectories – again by sweeping the time-delays and using nonlinear least squares fitting to the numerical derivatives. This procedure gave us a thousand sets of parameters for each model which corresponded to the same data.

### ***Real and Simulated Perturbations***

This section verifies that the model produces similar effects when perturbed to wet experiments. As described above, we collected data on a wild and mutated strain of *B. subtilis*. The genetically modified strain increased the production of *ComS*. This resulted in longer duration competence events and increased variability in competence events.

We first tuned the model to the experimental data of the wild type data. We then simulated the model in Matlab with lightly varied parameters to resemble small variance among cells of the same type.

Next, we increased the parameters which correspond to production of *ComS*: *alpha-S*, *beta-S*, and *k-S*. This is done to mimic the change in the modified *B. subtilis* strain.

In Figure 15.10, we show the side by side comparison between the simulated effects on the model and the experimental modified strain. The model does not show increased variance because it is a deterministic differential equation model and does not model the low-level stochastic nature of the system. However, the model predicts the same effect on competence durations as in experiment. The durations increased, and the normalized conserved quantity value increased.

**Finding Symbolic Parameters**

The search over equation space produces equations with bulk parameters; however, we can use a second equation search to identify the fully parameterized equation with symbolic parameters such as lengths, masses, etc. For example, in Chapter 11 our method found the following equation for the double pendulum with bulk parameters:

$$k_1\omega_1^2 + k_2\omega_2^2 + k_3\omega_1\omega_2 \cos(\theta_1 - \theta_2) - k_4 \cos \theta_1 - k_5 \cos \theta_2$$

The question is what are the symbolic representations for the  $k_i$  coefficients? To find the fully parameterized equation, we simply need data from similar systems but with different physical configurations and hence varying bulk parameters – for example, collecting data from several double pendula that have different arm lengths and masses.

One way to help identify the units in a potential invariant equation is to require the evolved expressions to be consistent in physical units, and to provide the algorithm with physically-meaningful building blocks such as the masses and lengths of the system's components, while requiring all other constants to remain unit-less. This approach still does not eliminate completely some fundamental ambiguities.

Alternatively, once we have found the invariant equation with bulk coefficients, we can refit it very easily to data from another system that has different parameters. If we do this on several different system configurations, we can obtain bulk coefficients for each configuration of the system versus the physical parameters (e.g.  $k_i$  values versus *length* and *mass* values of the collection of systems).

With bulk coefficient values from several systems, we can now find an equation for

each individual coefficient using explicit symbolic regression (e.g. find the equation of  $k_i$  as a function of the system masses and lengths).

We have done this in silico using 100 simulated double pendula with random masses and arm lengths. We first collected data from these double pendula by simulating them numerically and then refitting the coefficients of the double-pendulum equation for each. Since the partial derivative pairs metric is scale invariant, we divide out the first coefficient to put all equation in a normal form. This allows us to compare coefficients across multiple double pendulum equations. Finally, we use explicit symbolic regression to find the equation for each coefficient:

$$k_1/k_1 = 1$$

$$k_2/k_1 = m_2 L_2^2 / (m_1 L_1^2 + m_2 L_1^2)$$

$$k_3/k_1 = 2.00055 m_2 L_2 / (m_1 L_1 + m_2 L_1)$$

$$k_4/k_1 = 19.6 / L_1$$

$$k_5/k_1 = 19.6 \cdot m_2 L_2 / (m_2 L_1^2 + m_1 L_1^2)$$

where  $m_1$ ,  $L_1$ ,  $m_2$ , and  $L_2$  are the masses and lengths of the first and second arms respectively. The remaining coefficient 19.6 is a multiple of the gravitational acceleration 9.8 m/s (which we do not vary).

By multiplying the coefficients by their common denominator  $m_1 L_1^2 + m_2 L_1^2$ , we can finally write out the fully parameterized equation for arbitrary double pendula:

$$\begin{aligned} &L_1^2 (m_1 + m_2) \omega_1^2 + m_2 L_2^2 \omega_2^2 + 2 \cdot m_2 L_1 L_2 \omega_1 \omega_2 \cos(\theta_1 - \theta_2) \\ &- 19.6 \cdot L_1 (m_1 + m_2) \cos \theta_1 - 19.6 m_2 L_2 \cos \theta_2 \end{aligned}$$

Finding explicit equations for the parameters is much simpler than finding equations from scratch. Symbolic regression found each coefficient expression in less than 30 seconds, compared with the tens of hours required to find the original bulk coefficient equation.

## SECTION IV – APPLICATIONS

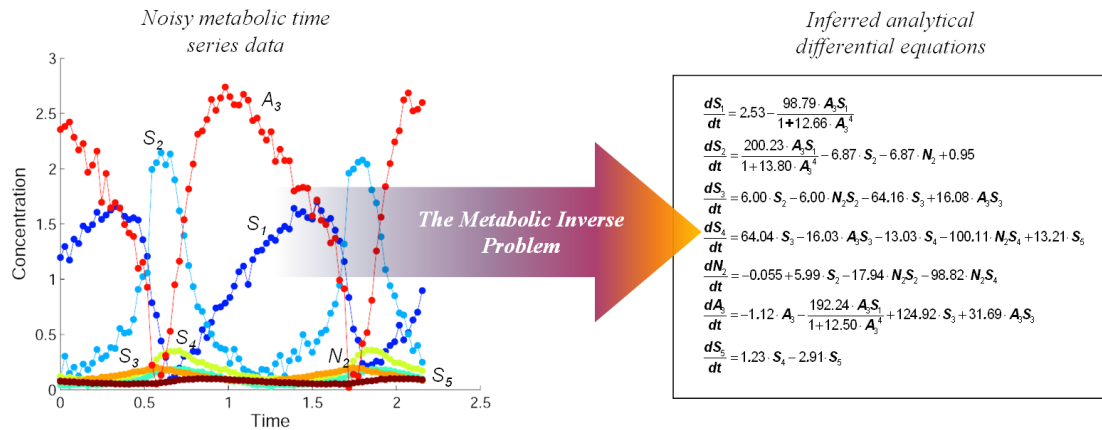
### CHAPTER 17. METABOLIC NETWORKS

#### **Summary**

Many challenges of systems biology involve reverse-engineering metabolic networks by using experimental data to determine metabolic fluxes. Traditionally, specification of the form of the analytical mathematical model appropriate to a particular metabolic system relies heavily on prior knowledge about the system, the experimental design, and how closely the system relates to established metabolic models. Here, we propose an automated process to build mathematical models with limited prior knowledge, or alternatively, adapt the form of a hypothesized model to suggest a more accurate structure. The algorithm alternates between generating multiple potential models commensurate with experimental data and designing new experiments that are optimized to differentiate models based upon disagreements between their predictions. We demonstrate the algorithm's ability on a noisy seven-dimensional model of yeast glycolytic oscillations and compare its performance with related methods. We further show that this method can symbolically correct impaired and overspecified expert models. We suggest that this approach may help study dynamic and non-linear components of complex metabolic and signaling systems, and may even provide optimized design and control of experiments in real-time.

#### **Introduction**

Many remarkable behaviors in nature arise from complex signaling or metabolic networks, and hence the ability to rapidly develop a predictive network model is essential to understanding and controlling these behaviors. A mathematical description is one way to represent the dynamics of a network amenable to human interpretation,



**Figure 17.1 Automated analytical modeling: Noisy time series data reflecting anaerobic metabolism concentrations over time are automatically translated into a set of coupled analytical differential equations without prior knowledge of the system (actual data and equations).**

but finding a full analytical expression can be arduous – particularly in multidimensional systems with nonlinear reactions, feedback, and oscillations that are common in biology. Here we propose a method that generates such a model *automatically* () without any *prior knowledge* of the metabolic system under study. It can be applied either to existing time-series data or wet-lab experiments suggested (or controlled) by the algorithm.

Identifying metabolic and signaling network models is of pressing practical interest (Stolovitzky and Califano 2007). A variety of methods have been used to infer gene regulatory networks (GRN) (Gardner, di Bernardo et al. 2003; Styczynski and Stephanopoulos 2005), including genetics, biochemistry, and molecular biology (Levine, Hu et al. 2007). Most often, preexisting models are used to provide a functional form, and then an optimization technique is used to fit the model parameters. Because of the breadth of data available, much of signaling network inference is based upon high-throughput mRNA microarray data for gene arrays, while metabolic network analysis considers both gene expression and high-throughput



mass spectrometry of metabolites (Nielsen and Oliver 2005). There are various challenges specific to the inference of metabolic networks from such data (Nemenman, Escola et al. 2007), since metabolism includes not only transcriptional regulation of enzymes, but also the conversion of substrate species with stoichiometric constraints. The computational challenge is exacerbated by the range of metabolic time constants and concentrations, which can easily span a several orders of magnitude, respectively.

While there remain many unsolved problems in the inference of GRN models, metabolic networks surpass many other biological networks in terms of their breadth, detail, quantitative nature, and experimental validation. Currently, it is possible to obtain quantitative, dynamic measurements of metabolic concentrations, metabolite fluxes, and genetic modification simultaneously, providing an important connection between the transcriptome/proteome and cellular phenotype (Ni and Savageau 1996; Kauffman, Pajerowski et al. 2002).

The most common mathematical form used to describe a metabolic network is a set of ordinary differential equations (ODEs) that describe the time-derivatives of chemical concentrations in the system as a function of its current state. ODEs are amenable to human interpretation because they are deterministic models and explicitly encode *causal relationships* (Bansal, Belcastro et al. 2007), including feedback loops that are difficult to model using other methods. Terms in the differential equations correspond to reactions occurring in the system based on their connectivity, such as first- and second-order rate laws, power laws, and Michaelis-Menten kinetics (Koza 2001).

Methods such as symbolic regression (Koza 1992; Augusto and Barbosa 2000; Duffy and Engle-Warnick 2002; Hoai, McKay et al. 2002) can be used to identify differential

equations automatically from experimental data (Schmidt and Lipson 2006; Bongard and Lipson 2007; Schmidt and Lipson 2007), however, substantial challenges remain to scale into the dimensionality and functional complexity necessary for biological applications.

In this chapter, we introduce a method to automatically construct mathematical models of a biological system, and apply this technique to infer a seven-dimensional nonlinear model of glycolytic oscillation in yeast – the largest automatically identified system to date – using only noisy observational data *in silico*. This method is enabled by three new techniques for searching for differential equation models: graph-based symbolic encoding (Schmidt and Lipson 2007), fitness prediction (Schmidt and Lipson 2006; Schmidt and Lipson 2008), and estimation-exploration (Bongard and Lipson 2005; Zykov, Bongard et al. 2005; Bongard and Lipson 2007).

## **Background**

### ***Metabolic Modeling***

Given the breadth of metabolic networks, we find it useful to classify systems biology metabolic models into three categories: comprehensive (exact and complete) versus local (surrogates or approximations), static versus dynamic, and linear versus non-linear. Genome-scale modeling using generalized mass action (Jamshidi and Palsson 2008) is linear, dynamic, and comprehensive. Flux balance analysis (FBA) and metabolic control analysis (MCA) are linear, static, and comprehensive. Metabolic flux analysis (MFA) is linear, static, and localized (Varma and Palsson 1994). Dynamic flux balance analysis (dFBA) (Mahadevan, Edwards et al. 2002; Gadkar, Varner et al. 2005) and dynamic metabolic control analysis (MCA) (Fell 1992; Mendes and Kell 1996; Kell 2004) are linear, static, and fall between localized and

comprehensive. Biochemical systems theory, also known as the S-System approach, is nonlinear, dynamic, localized (Beard, Qian et al. 2004; Crampin, Schnell et al. 2004). Cybernetic modeling is nonlinear, dynamic, and falls between localized and comprehensive (Young and Ramkrishna 2007). It is becoming more widely recognized that highly detailed comprehensive models suffer from the identifiability problem (Schmidt, Madsen et al. 2008), because of the inability to distinguish experimentally between parameter combinations that produce identical measurements, and that additional methods are needed to reduce model complexity. We focus this chapter on an approach to identify local or effective models for non-linear and dynamic subsets of larger systems, and hence explore the underlying physiology and enable external control of the system and the optimized design of wet-lab experiments.

Metabolic models, in contrast to signaling ones, require strict adherence to the stoichiometry of the equations, *i.e.*, chemical mass balance. Such mathematical models can be used to predict the behavior of the network in different conditions, such as attracting basins and bifurcations – predictions that are not readily available in statistical models. Stoichiometric methods can also be used to identify some qualitative properties of biological systems. For example, if a model can be linearized, it is possible to create a Jacobian matrix that can subsequently be decomposed into stoichiometric and gradient matrices to reveal kinetic and thermodynamic components (Jamshidi and Palsson 2008), but this technique may not be applicable to problems that are not readily linearized or for which perturbations take the system far from the reference model.

Integration of a parameterized system of differential equations is known as the forward metabolic network problem. In contrast, the inverse problem involves determining the nature of the equation network underlying observed behavior using

techniques such as reverse engineering or systems identification. Reverse-engineering a metabolic network consists of determining both the correct functional *form* of a set of ODEs to describe the system and the proper set of model parameters to fit experimentally collected data to within a given tolerance. The inverse metabolic problem is universally recognized as very hard (Kell 2004; Kell 2006) and most likely NP complete (Mendes and Kell 1996; Styczynski and Stephanopoulos 2005). As a result of the nonlinear and coupled nature of the equations, enzymatic kinetics studied in isolation or with small, singular perturbations, often used to explore network connectivity, may not be informative regarding the behavior of the complete system, particularly under large amplitude dynamic perturbations to multiple variables. Conventional local nonlinear solvers can be inadequate for the ill-conditioned and multi-model inverse problem presented by the nonlinear, differential-algebraic constraints associated with dynamic biochemical pathways, and various global nonlinear optimization approaches have been developed to solve the inverse problem (Mendes and Kell 1998; Moles, Mendes et al. 2003; Beard, Qian et al. 2004; Crampin, Schnell et al. 2004).

## **Methods**

### ***Searching for Differential Equations***

Genetic programming is a widely studied class of evolutionary algorithms inspired by biological evolution (Koza 1992). In a traditional genetic program, an initially random population of solutions evolves iteratively in computer memory to maximize some objective – for example, to model experimental data with the lowest squared error. Solutions with the highest fitness persist in the population to *recombine* (genetic crossover) and *mutate* to replace less fit individuals.

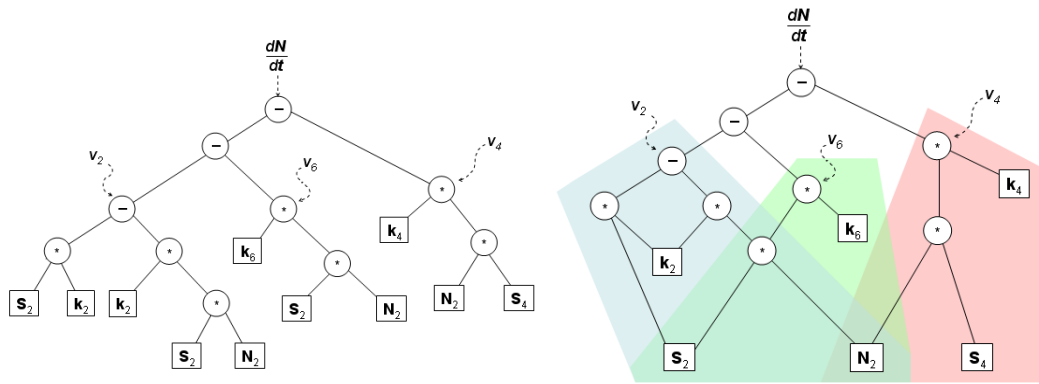
Symbolic regression uses *genetic programming* to evolve (compete) algebraic expressions to explain experimental data (Koza 1992). Unlike polynomial regression or neural networks which also fit data, symbolic regression searches a space of analytical equations to explain experimental observations. Symbolic regression composes equations using basic algebraic building blocks with the aim to formulate simpler (*e.g.*, fewer parameters) or more natural expressions (robust to perturbations) that are more likely to correspond to the underlying intrinsic behavior mechanisms of the system.

Symbolic regression compares candidate equations by calculating their residual errors on the experimental data – also known as the equations *fitness metric* – for example, using square-error or correlation. In past research, algorithms have used all available data at once to evaluate the fit. However, this metric can be overly stringent and inhibit solutions from building intermediate expressions needed for the final model.

Instead, we use the technique *fitness prediction* to reduce overall computational cost and to improve the local search gradient (Schmidt and Lipson 2006; Schmidt and Lipson 2008). Fitness predictors measure error on only a small subset of the data. The data subset is adapted, however, as a population of fitness predictors (data subsets) evolves in parallel with symbolic regression of differential equations. Predictors are rewarded for accurately approximating many equations' error on the full data set. All differential equations measure fitness using the top-ranked predictor. In contrast to standard symbolic regression, equations compete on an accurate fitness approximation but are free to drift in more trajectories. Predictors adapt to defeat poor deviations.

Conceptually, fitness prediction allows a genetic algorithm to search a wider range of solutions by adapting the fitness heuristic and reducing its computational cost. An

$$\frac{dN}{dt} = k_2 S_2 - k_2 S_2 N_2 - k_6 S_2 N_2 - k_4 S_4 N_2$$



**Figure 17.2.** Analytical model representations for NADH in the cell glycolysis model - a tree encoding (left pane) and a graph encoding (right pane). Both panes encode the same equation, but while the tree encoding is simpler to manipulate algorithmically (*e.g.*, alter subexpressions), it requires redundant subtrees and is prone to produce large equations that may not accurately represent the biological system. The graph encoding couples subtrees, thereby biasing equations to preserve simpler shared expressions.

interesting result (Schmidt and Lipson 2008) shows that symbolic regression is substantially more successful when solutions are pressured to explain only a few features of the systems at any given time rather than the entire data set at once. This allows solutions to drift from the objective gradient, but the focus adapts with the solution population to prevent excessive divergence from the intended gradient.

### ***Model Encoding***

The ability to identify an accurate and parsimonious differential equation model using symbolic regression relies critically on the *genetic encoding* (*e.g.*, the genotype organization of a symbolic expression). To search the space of candidate symbolic analytical equations, we use an acyclic graph encoding for symbolic regression that scales well computationally and exploits the shared structures found in many metabolic networks (Schmidt and Lipson 2007). Traditionally, symbolic expressions

have been represented as binary-trees, where parent nodes represent algebraic operations such as addition or multiplication, and leaf nodes represent symbolic variables and parameter constants (Figure 17.2A, left pane). However, trees can produce complex and bloated equations, often resulting in unsuitable models for understanding the underlying system. Instead, the graph encoding produces models that are more concise on average by reusing and coupling sub-expressions in the genetic encoding (Schmidt and Lipson 2008).

The acyclic graph encoding represents a symbolic expression by interpreting nodes as mathematical operations such as addition and multiplication. Leaf nodes represent state-variables or parameter constants (Figure 17.2B, right pane). The encoding for the graph is an ordered list of operations much like assembly code: Each operation builds up successive sub-expressions in the final expression, using any preceding operations and symbolic variables. The graph encoding takes advantage of redundant sub-expressions, such as coupled reactions in metabolic networks, and is biased against bloated solutions and overfitting (Schmidt and Lipson 2007).

The acyclic graph (illustrated in Figure 17.2) that represents symbolic equations was encoded internally as floating-point assembly code. The encoding consists of a list of floating-point operations and parameter values. Operations can load an input variable or a parameter value (*set* command), or perform a floating-point operation on any previous operation outputs (*add/sub/multiply/divide* commands). Each operation corresponds to a leaf or parent node in the graph. The graph is rooted by the final operation in the list. Table 17.1 shows several raw encodings generated by the algorithm after regressing the yeast glycolysis model.

**Table 17.1. Raw encodings of glycolysis differential equations found.**

$S_1$	$S_2$	$S_3$	$S_4$
(0) ← set <A3> (1) ← set [-7.15469] (2) ← set <S1> (3) ← mul (1) (2) (4) ← set [-10.6171] (6) ← div (4) (3) (10) ← set <S1> (12) ← set <S3> (13) ← div (3) (12) (15) ← sub (6) (10) (16) ← div (13) (15) (17) ← sub (16) (0) (18) ← sub (16) (17) (22) ← mul (18) (18) (23) ← set [0.07081] (24) ← div (23) (0) (25) ← mul (18) (22) (26) ← add (24) (25) (27) ← div (3) (26) (28) ← set [-2.469] (31) ← sub (27) (28) return (31)	(0) ← set [-0.2349] (1) ← set [-6.00913] (2) ← set <S2> (3) ← mul (1) (2) (4) ← add (0) (3) (5) ← set [-6.70044] (7) ← mul (5) (2) (8) ← set <N2> (9) ← mul (7) (8) (10) ← add (4) (9) (11) ← set [14.6053] (12) ← set <S1> (13) ← mul (11) (12) (14) ← set [0.0710] (15) ← set <A3> (16) ← div (14) (15) (19) ← mul (15) (15) (21) ← mul (19) (15) (22) ← add (16) (21) (23) ← div (13) (22) (24) ← add (10) (23) (25) ← set [-0.1942] (26) ← add (24) (25) (27) ← set [-0.4663] (28) ← sub (26) (27) (29) ← set [1.01609] (31) ← div (28) (29) return (31)	(0) ← set [6.01392] (1) ← set <S2> (2) ← mul (0) (1) (3) ← set [-64.187] (4) ← set <S3> (5) ← mul (3) (4) (6) ← add (2) (5) (7) ← set [16.0479] (9) ← mul (7) (4) (10) ← set <A3> (11) ← mul (9) (10) (12) ← add (6) (11) (13) ← set [-6.0004] (14) ← set <S2> (15) ← mul (13) (14) (16) ← set <N2> (17) ← mul (15) (16) (28) ← add (12) (17) (29) ← set [1] (31) ← div (28) (29) return (31)	(0) ← set [-0.02674] (1) ← set [62.8684] (2) ← set <S3> (3) ← mul (1) (2) (4) ← add (3) (0) (5) ← set [-12.727] (6) ← set <S4> (7) ← mul (5) (6) (8) ← add (4) (7) (9) ← set [12.7542] (10) ← set <S5> (11) ← mul (9) (10) (12) ← add (8) (11) (13) ← set [-98.402] (15) ← mul (13) (6) (16) ← set <N2> (17) ← mul (15) (16) (18) ← add (12) (17) (19) ← set [-15.712] (20) ← set <S3> (21) ← mul (19) (20) (22) ← set <A3> (23) ← mul (21) (22) (24) ← add (18) (23) (25) ← set [1.01302] (26) ← mul (24) (25) (27) ← set [1.00701] (28) ← mul (26) (27) (29) ← set [0.0213] (31) ← add (28) (29) return (31)
$N_2$	$A_3$	$S_5$	
(1) ← set [5.95097] (2) ← set <S2> (3) ← mul (1) (2) (5) ← set [-17.8537] (6) ← set <S2> (7) ← mul (5) (6) (8) ← set <N2> (9) ← mul (7) (8) (10) ← add (3) (9) (11) ← set [-99.130] (12) ← set <S4> (13) ← mul (11) (12) (15) ← mul (13) (8) (16) ← add (10) (15) (17) ← set [0.9840] (18) ← mul (16) (17) (19) ← set [0.9841] (20) ← div (18) (19) (27) ← set [-0.0003] (28) ← add (20) (27) (29) ← set [1.01106] (31) ← mul (28) (29) return (31)	(0) ← set [0.08596] (1) ← set [128.854] (2) ← set <S3> (3) ← mul (1) (2) (4) ← add (0) (3) (5) ← set [-1.37961] (6) ← set <A3> (7) ← mul (5) (6) (8) ← add (4) (7) (9) ← set [-32.0337] (11) ← mul (9) (2) (13) ← mul (11) (6) (14) ← add (8) (13) (15) ← set [-14.53] (16) ← set <S1> (17) ← mul (15) (16) (18) ← set [0.0714] (19) ← set <A3> (20) ← div (18) (6) (23) ← mul (6) (6) (25) ← mul (23) (19) (26) ← add (20) (25) (27) ← div (17) (26) (28) ← add (14) (27) (29) ← set [0.99359] (31) ← mul (28) (29) return (31)	(0) ← set [1.30265] (1) ← set <S4> (2) ← mul (1) (0) (3) ← set [-3.1032] (4) ← set <S5> (5) ← mul (3) (4) (6) ← add (2) (5) (25) ← set [-2265.4] (26) ← add (6) (25) (28) ← sub (26) (25) (29) ← set [-0.0001] (31) ← add (28) (29) return (31)	



The connected components of the graph define a sequence of operations that correspond to a single equation, as shown in Table 17.1.

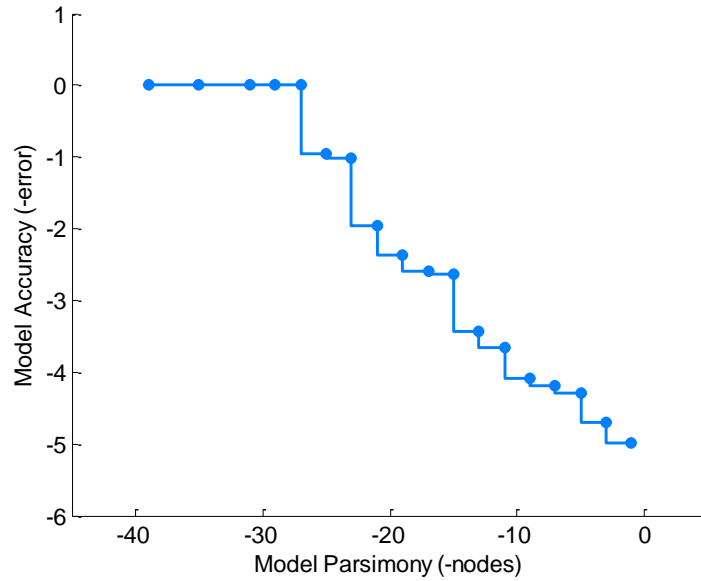
In our experiments, we are effectively searching the rational functions (seven-variable quotients of polynomials) of at most 32 operations (nodes in an acyclic graph representation). This places a limit on the total number of parameters also to 32. The discrete search space size, neglecting real-valued parameters, is thus  $6^{32}$  – or roughly  $10^{25}$  parameterized functions.

### ***Model Accuracy and Complexity Tradeoff***

For any given system, there a potentially infinite set of equations that closely fit any finite set of experimentally collected data. Therefore, it is important to have some qualitative understanding of what the domain of reaction rate equations looks like. For example, a 1000<sup>th</sup> order polynomial can perfectly fit any data set of 1000 or fewer unique time samples. Therefore, it is important to understand the qualitative features of the equation-space which can also help us distinguish between true intrinsic models and coincidental fits.

Consider the relationship between equation complexity and accuracy of fitting the experimental data. Qualitatively, there exist extremely complex equations (*e.g.*, Taylor series, neural networks, and Fourier series) with near perfect accuracy as well as simple, single-parameter models with baseline accuracy (*e.g.*, the mean reaction rate). The behavior of equations in between these two extremes is more interesting.

Figure 17.3 shows the Pareto front of equation accuracy versus equation complexity for modeling a particular reaction rate ( $dS_1/dt$  described below). It demonstrates a cliff point in the trade-off between model accuracy and complexity. Starting at the lower right corner of the figure and increasing the model complexity by moving to the left,



**Figure 17.3. The pareto front of model accuracy versus its simplicity. There is an inherent trade-off between complexity and accuracy to the training data. Many complex functions have very high accuracy, however the exact solution lies at the sharp inflection near 28 nodes, balancing high accuracy and simplicity.**

there is a certain complexity where model accuracy jumps dramatically and then plateaus. In other words, there is a relatively simple equation that can model the system's behavior accurately (but perhaps not perfectly). By parsimony arguments, we can reason this equation to be the most-likely model of the system. The equation at the inflection at this example is indeed the correct  $S_1$  model, supporting this assumption.

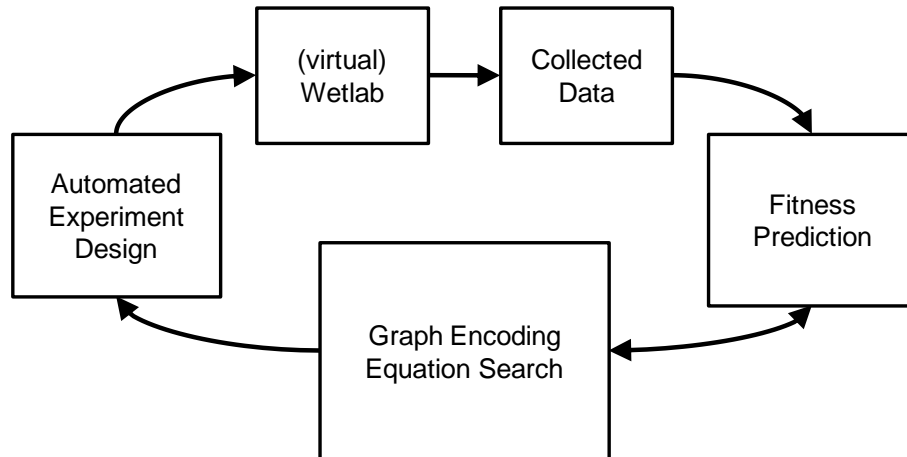
### *Automated Experimental Design*

Once the symbolic regression step has evolved a population of candidate solutions to fit the current set of training data, there may be several coherent solutions for modeling the data in different ways – particularly in high-dimensional domains with sparse data where many equivalent explanations exist for the simplest behavior. But which mathematical explanation of the system is correct? The estimation-exploration algorithm (EEA) is a method to automatically design a new experiment that can help differentiate the current solution candidates and refine their structure (Bongard and

Lipson 2005; Zykov, Bongard et al. 2005; Schmidt and Lipson 2006; Bongard and Lipson 2007). The purpose of the EEA is to decipher which model is likely to be correct by searching for experiment settings, perturbations, or procedures that cause current models to disagree most in their predictions. Figure 17.4 summarizes the high-level symbolic regression of differential equations and the automated experiment control of the proposed algorithm.

The first step in our exploration of an “unknown” metabolic network is to perform a series of randomly selected experiments – perhaps just observing nominal stable behavior, such as stable nodes and limit cycles. As candidate solutions compete to fit these training data, there is a tendency to produce multiple solutions that explain the behavior in different ways. Given multiple solutions competing to explain the current data, we can then search in parallel for new experiment designs to maximize disagreement in the predictions of the set of solutions. For a dynamical system such as glycolysis, we design new experiments as sets of initial conditions into which we place the system and then record its transient trajectory as governed by the differential equations in the black box. We dictate the most informative experiment to be the set of initial conditions in which the current population of solutions has the highest statistical variance in its predicted dynamics. The candidate experiment producing the most disagreement in the prediction of competing models is the most informative experiment to carry out and the one most likely to eliminate overfit models that are unable to make useful predictions (Zykov, et al., 2005).

Once identified by the EEA, we can then perform the most controversial experiment on the real system, acquire new data, and once again compete solutions to explain them. We repeat this process (Bongard and Lipson 2005; Zykov, Bongard et al. 2005; Bongard and Lipson 2007) until a single dominant solution emerges.



**Figure 17.4** The coevolution of models through symbolic regression and fitness prediction, and experiments by the estimation-exploration algorithm. Candidate solutions compete to explain current experimental data, and experimental initial conditions compete to maximize disagreement in the predictions of the various solutions. This process of synthesizing coherent models and controversial experiments continues until a single dominant solution emerges.

### *Distributed Computation*

Genetic programs are readily parallelizable to several computers and server clusters where available. We distributed the symbolic regression evolution over four computers and eight total logical processors using the island distributed computation method (Francisco, Marco et al. 2003). The island model partitions the population of solutions into separated smaller populations residing on each computer (or core). We spread a population of 512 individuals over eight CPU cores; therefore each population has 64 individuals.

The island model populations are faster to evolve because there are fewer individuals and less work to calculate fitness values per population. We migrate solutions between populations at regular intervals. Every 10,000 iterations (averaged over all populations), we randomly shuffle all solutions among random pairs of populations.

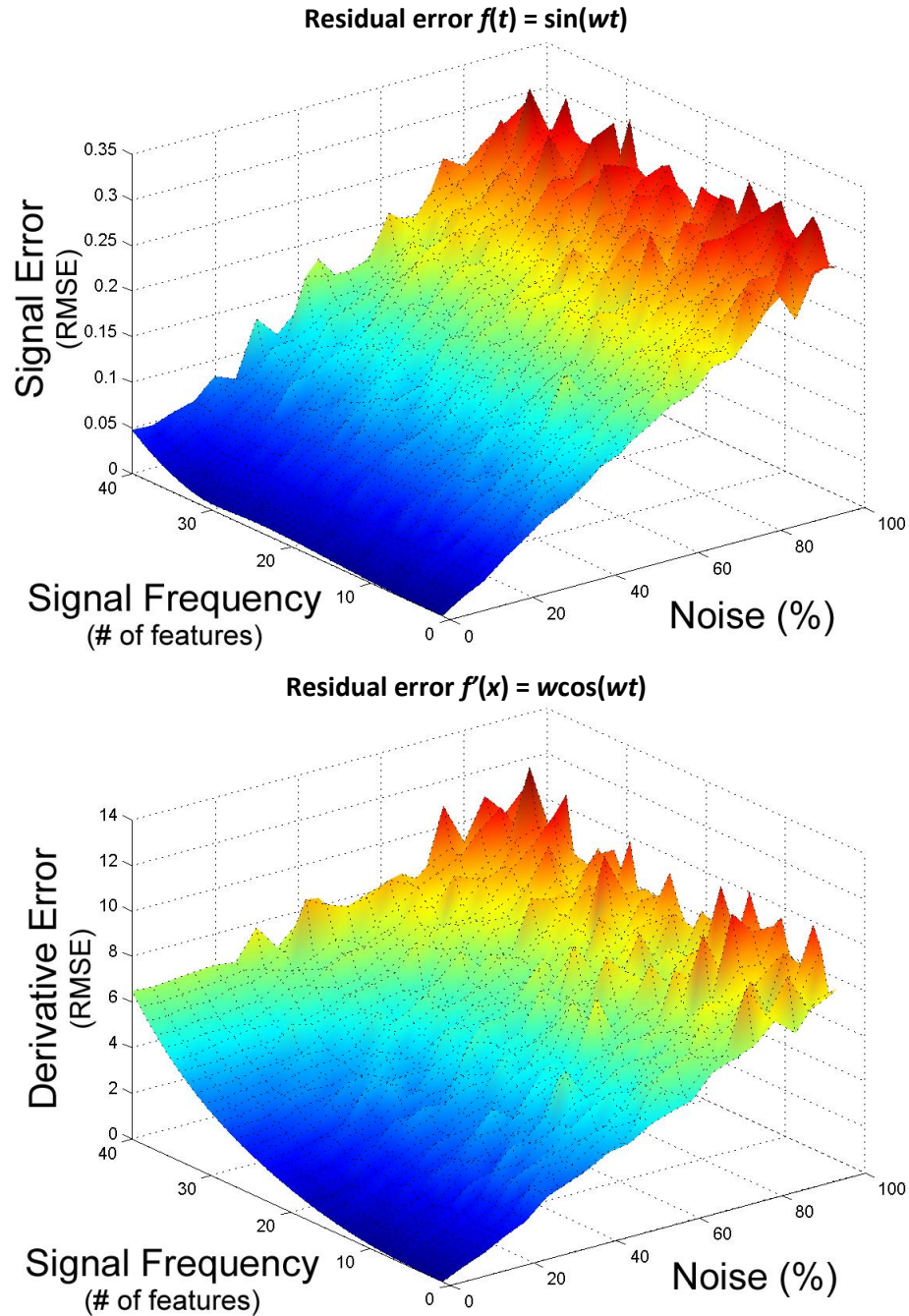
### *Noise Effects on Numerical Derivatives*

Measurement noise makes approximating the gradient (numerical derivatives) more difficult because derivatives can be highly sensitive to noise. We used non-parametric fitting, Loess smoothing (Cleveland and Devlin 1988), which could overcome a significant amount of noise, up to a point depending on the noise strength and frequency.

Loess smoothing updates each sample in the data set by fitting a small order polynomial to the sample and its nearest neighbors. If the neighbor size is significantly wider than the sample rate, the polynomial will remove high-frequency noise. Other methods, such as filtering and convolution, also reduce high-frequency noise, but they do not readily produce estimates of the signal derivative. Using Loess smoothing, we can obtain the numerical derivative directly from the smoothing procedure by evaluating the symbolic derivative of the local polynomial fit at each data sample.

In Figure 17.5, we can see the effect of Loess smoothing for calculating the numerical derivative versus the amplitude of the noise and its frequency relative to the sampling rate. These graphs come from smoothing the signal  $f(t)=\sin(\omega t)$  over  $t=[0,2\pi]$ . The number of features (of the data set) is defined as  $2\pi\omega$  (the number of periods in the data set). We can see that error on the signal itself is most affected by the noise frequency. In contrast, the error of the numerical derivative using Loess smoothing is affected by both noise amplitude and the number of features in the data set (frequency of the signal).

This result suggests that smoothing cannot remove all noise from data, even for small amounts, and that smoothing breaks down for the numerical derivative values for high-frequency features in the data.



**Figure 17.5.** The residual squared-error after Loess smoothing versus the magnitude of the noise and the density of features relative to the noise frequency (sample rate) for a sine-wave signal and its numerical derivative. The signal error is most sensitive to the noise magnitude but more robust to the number of features. In contrast, the error on the numerical derivative has much higher sensitivity to the number of features. The state of the art of what the symbolic regression algorithm can handle with Loess smoothing is roughly the medium-blue to dark-blue regions.

### ***The Glycolytic Oscillation Models***

We begin with a published numerical model (Wolf and Heinrich 2000; Ruoff, Christensen et al. 2003) of glycolytic oscillation in yeast for the system upon which our algorithm experiments. Table 17.2 and Table 17.3 provide details of the models shown in Figure 17.6.

In this seven-variable model, the respiratory chain (mitochondrial oxidative phosphorylation) is completely inhibited. The reaction network for this system, shown in Figure 17.6 (left), contains the main reactions of glycolysis and adjacent reactions producing ethanol and glycerol. In Table 17.2 we list the chemical species and their rate/mass balance equations and initial conditions, and in Table 17.3 the associated reaction fluxes and kinetic coefficients. During model development, the complexity of the model was reduced by omitting many of the glycolytic reactions, and by lumping together other reactions, so that several of the model variables denote concentrations of pools of intermediates rather than concentrations of the individual compounds, *e.g.*, the pools of triose phosphates (glyceraldehydes-3-phosphate, dihydroxyacetone phosphate) and pyruvate and acetaldehyde. This simplification has been rigorously justified using a judiciously applied quasi steady-state approximation (Heinrich, Rapoport et al. 1977).

This particular model is capable of reproducing glycolytic oscillations with a period in the range of 0.10 to 12 min and has been used to study the temperature dependence and temperature compensation of yeast glycolytic oscillations (Ruoff, Christensen et al. 2003).

**Table 17.2. The chemical species in the model (NM, IM, and OS are the normal, impaired, and overspecified models, respectively).**

Variable	Description	Model	Species rate or mass balance	Initial conditions
$A_2$	ADP	All	$A_2 + A_3 = A$	1.525 mM
$A_3$	ATP	All	$\dot{A}_3 = -2v_1 + 2v_3 - v_5$	2.475 mM
$N_1$	NAD <sup>+</sup>	All	$N_1 + N_2 = N$	0.923 mM
$N_2$	NADH	NM OS	$\dot{N}_2 = v_2 - v_4 - v_6$	0.077 mM
		IM	$\dot{N}_2 = v_2 - J_P$	
$S_1$	Glucose	All	$\dot{S}_1 = J_G - v_1$	1.187 mM
$S_2$	Glyceraldehydes-3-phosphate and dihydroxyacetone phosphate pool	NM OS	$\dot{S}_2 = 2v_1 - v_2 - v_6$	0.193 mM
		IM	$\dot{S}_2 = 2v_1 - v_2$	
$S_3$	1,3-bisphosphoglycerate	All	$\dot{S}_3 = v_2 - v_3$	0.050 mM
$S_4$	Cytosolic pyruvate and acetaldehyde pool	NM	$\dot{S}_4 = v_3 - v_4 - J_P$	0.115 mM
		IM	$\dot{S}_4 = v_3 - J_P$	
		OS	$\dot{S}_4 = v_3 - v_4 - J_P - v_{sink}$	
$S_5$	Extracellular concentration of $S_4$	All	$\dot{S}_5 = \varphi(J_P - v_7)$	0.077 mM $\varphi = 0.10$

We simulate collecting wet-lab experimental data by adding noise sampled from the normal distribution to each state-variable in the time-series. Each state measurement is given 10% noise (where the standard deviation of the noise is 10% of the standard deviation of the corresponding state-variable in its stable cycle). We then calculate the derivatives of the resulting seven state-variables numerically using locally weighted polynomial fitting (Cleveland and Devlin 1988).



**Table 17.3. Description of the reaction fluxes and their kinetic coefficients**

Reaction enzymes or processes	Model	Reaction	Coefficient value
Incoming flux of glucose across cell membrane	All	$J_G = \text{constant}$	$J_G = 2.5 \text{ mM/min}$
Hexokinase, phosphoglucosomerase, and phosphofruktokinase, where $K_I$ is the inhibition constant and the exponent 'q' is the cooperativity coefficient of ATP inhibition	All	$v_1 = \frac{k_1 S_1 A_3}{1 + \left(\frac{A_3}{K_I}\right)^q}$	$k_1 = 100 \text{ mM/min}$ $K_I = 0.52 \text{ mM}$ $q = 4.0$
Glyceraldehydes-3-phosphate dehydrogenase	All	$v_2 = k_2 S_2 N_1$	$k_2 = 6.0 \text{ mM/min}$
Phosphoglycerate kinase, phosphoglycerate mutase, enolase, and pyruvate kinase	All	$v_3 = k_3 S_3 A_2$	$k_3 = 16.0 \text{ mM/min}$
Alcohol dehydrogenase	NM OS	$v_4 = k_4 S_4 N_2$	$k_4 = 100 \text{ mM/min}$
	IM	Absent	
Nonglycolytic ATP consumption	All	$v_5 = k_5 A_3$	$k_5 = 1.28 \text{ min}^{-1}$
Formation of glycerol from triose phosphates	NM OS	$v_6 = k_6 S_2 N_2$	$k_6 = 12.0 \text{ mM/min}$
	IM	Absent	
Degradation of pyruvate and acetaldehyde in the extracellular space	All	$v_7 = k S_5$	$k = 1.8 \text{ min}^{-1}$
Carbon sink term to the pyruvate pool accounting for the carbon loss to cellular synthetic processes (fatty acid biosynthesis, amino acid production)	OS	$v_{\text{sink}} = \frac{k_{\text{sink}} S_4}{1 + \left(\frac{A_3}{K_{IATP}}\right)^3}$	$k_{\text{sink}} = 20 \text{ mM/min}$ $K_{IATP} = 0.52 \text{ mM}$
Membrane transport of pyruvate and acetaldehyde into extracellular space ( $A_s$ = membrane surface, $P$ = membrane permeability, and $V$ = cellular volume)	NM OS	$J_P = \left(\frac{A_s P}{V}\right)(S_4 - S_5)$	$\left(\frac{A_s P}{V}\right) = 13.0 \text{ min}^{-1}$
	IM	$J_P = \left(\frac{A_s P}{V}\right)(S_4)^*$	

\* In the case of the impaired model, mammalian cells do not typically take in lactate from the extracellular space, so the dependence on  $S_5$  was eliminated to ensure that the model would act like a mammalian cell.

### ***Generating Data***

We generated data by numerically integrating the glycolysis model from an initial state and recording the state-variables over the transient trajectory. The initial state is either randomly chosen (for collecting initial data before regression) or chosen by the algorithm. For a given initial state, we record the trajectory every 0.1 minutes until we have acquired 100 samples. Given the approximately one-minute period of the limit cycle, this allows us to observe the transient behavior that occurs as the initial state progresses towards the limit cycle, but not redundantly sample the limit cycle for typical initial states. The algorithm runs a new experiment (perturbs an initial state and collects new data) every 50,000 iterations (roughly every 10 minutes). All initial states were confined to viable environments as indicated in Table 17.4. The upper-bound constraints are doubled for the test set to expand the phase space by a factor of  $2^7$  and measure how well models extrapolate and predict new behavior.

Once the data are generated, we simulate physical measurements by adding normally distributed random noise to each state-variable in each time-sample. The standard deviation of the random noise added to each state is relative to the standard deviation of the state-variable in the system's stable limit cycle. This gives variables with large magnitude oscillations higher noise than variables with smaller magnitudes. This also makes the noise independent of measurement units. We used 10% noise, *i.e.*, the ratio of the noise standard deviation to the variable standard deviation is 0.1.

We smooth the noisy time-series numerically using Loess locally weighted polynomial fitting (Cleveland and Devlin 1988) with window size of 50. Additionally, we approximate the derivative of the time-series by evaluating the derivative of the local polynomial fit of each point.

**Table 17.4. Model variables, the allowed range of initial states for the training data set, and the standard deviation of the limit cycle used to compute the amount of added noise.**

<b>Variable</b>	<b>Name</b>	<b>Range</b>	<b>Standard deviation</b>
$S_1$	Glucose	[0.15, 1.60]	0.4872
$S_2$	Glyceraldehydes-3-phosphate and dihydroxyacetone phosphate pool	[0.19, 2.16]	0.6263
$S_3$	1,3-bisphosphoglycerate	[0.04, 0.20]	0.0503
$S_4$	Cytosolic pyruvate and acetaldehyde pool	[0.10, 0.35]	0.0814
$N_2$	NADH	[0.08, 0.30]	0.0379
$A_3$	ATP	[0.14, 2.67]	0.7478
$S_5$	Extracellular pyruvate and acetaldehyde pool	[0.05, 0.10]	0.0159

### ***Symbolic Regression Algorithm Settings***

We use the fitness prediction symbolic regression algorithm described in (Schmidt and Lipson 2008) to build different equations to fit time-series data. We use a population

size of 512, distributed over eight CPUs/cores. We use the deterministic crowding selection method, with 5% mutation probability and 75% crossover probability. The encoding is an operation list acyclic graph with a maximum of 32 operations/nodes. Single-point crossover exchanges operations in the operation list at a random split. The operation set contains addition, subtraction, multiply, and divide algebraic operations.

The fitness predictor population contains 128 predictors, distributed over eight CPUs/cores. The fitness predictor subset size is 16 indices to the full training data set. Predictors are evolved via deterministic crowding, with 10% mutation and 50% crossover.

We calculate fitness using the correlation coefficient between the candidate solution's predicted derivative values and the numerically estimated derivatives from the training data. We also include a small absolute error term to provide a weak gradient to match the scale and offset of the data. The fitness function for a solution  $s$  is therefore:

$$fitness(s) = \frac{cov(x, y)}{\sigma_x \sigma_y} - \varepsilon \cdot \frac{1}{n} \sum |x - y|$$

where  $s$  is a candidate differential equation,  $x$  is the model's predicted derivative values,  $y$  is the numerical derivative from the training data,  $\sigma_x$  and  $\sigma_y$  are their respective standard deviations, and  $cov(x,y)$  is the covariance of  $x$  and  $y$ . The summation is the small mean-absolute-error term, with  $\varepsilon$  equal to  $10^{-6}$ . When calculating the exact fitness of a candidate solution,  $x$  and  $y$  values cover the entire training data. When predicting fitness, the  $x$  and  $y$  values cover only data samples referenced by the predictor.

### ***Regression Procedure***

During regression for each compared algorithm – symbolic regression, nonlinear regression, and neural network regression – we track accuracy on both the training and test data sets over time. Only the training data set is used to update the models. By recording the accuracy of the model of the test set over time as well, we can analyze later how well the regression procedure is generalizing the model to data not in the training set (e.g. Figure 17.12 and Figure 17.13). Additionally, we record performance on a third validation data set. The validation data set (same size and phase distribution as the training data) is used only to choose the best point during regression that maximizes generalization (a method known as “early-stopping”) for display in Figure 17.13.

In nonlinear regression and neural network regression, the training set was constant, with 200 trajectories as described earlier. In contrast, the symbolic regression algorithm’s training set begins with 10 random trajectories, but adds new trajectories chosen by the algorithm throughout regression. For all algorithms, the test data set was held constant. The test data set contained 100 random trajectories as described previously.

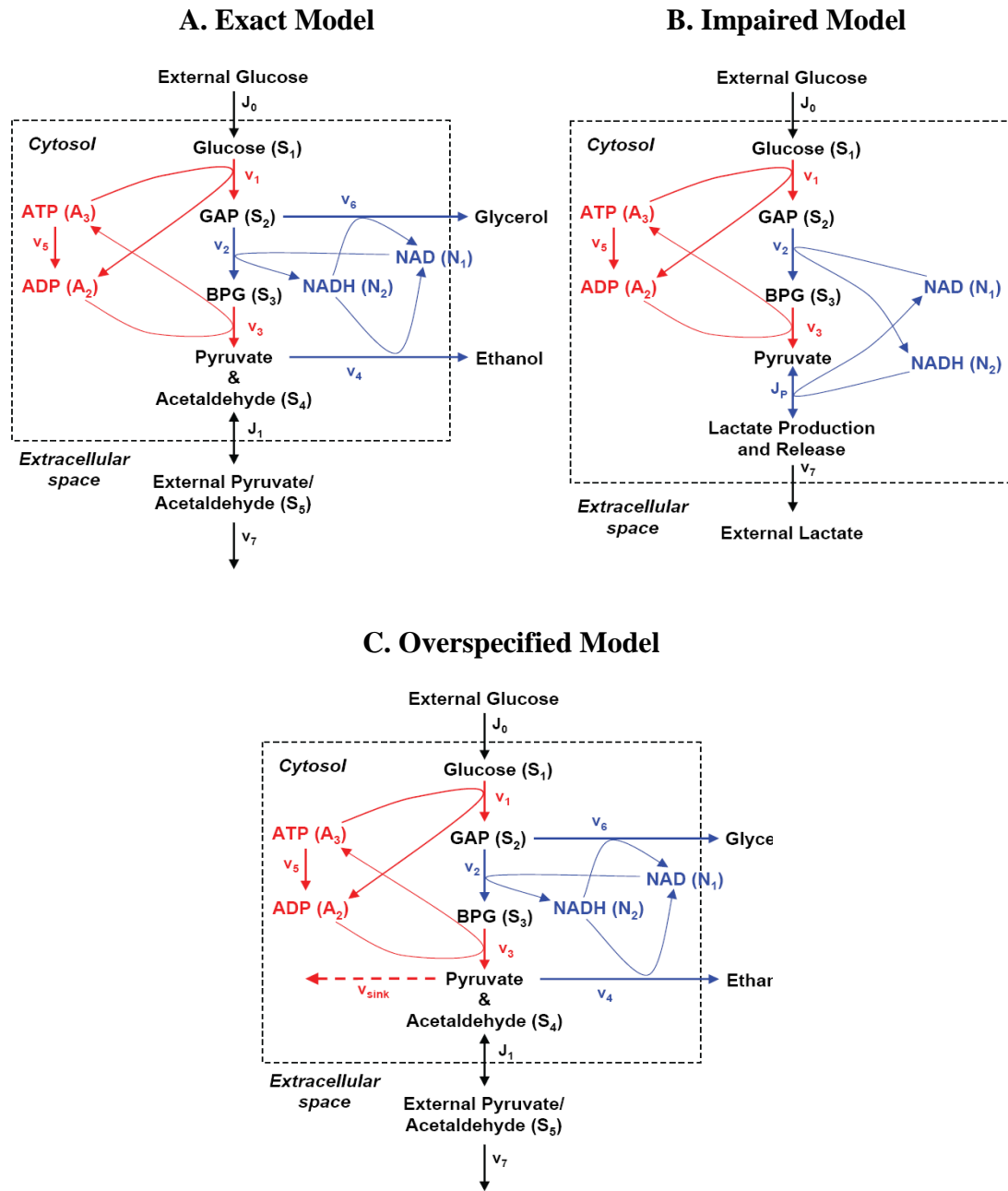
### **Results and Discussion**

We have applied this automated modeling procedure to the *in silico* analysis of a seven-dimensional model of glycolytic oscillations in yeast. We have tested the regression of the entire system *ab initio* (without any prior knowledge) and compared prediction results with nonlinear regression and neural nets. Finally, we determined the ability of the algorithm to adapt and correct a partially incorrect hypothesized model chosen by the experimenter to fit the exact system, *i.e.*, to augment expert modeling.

### ***Reverse-engineering Glycolytic Oscillations in Yeast***

We used the models of glycolytic oscillations in yeast shown in Figure 17.6A to simulate experimenting on a wet system. Glycolytic oscillation is one of the most common examples of oscillatory behavior at the cellular level and enables a broader understanding of the underlying dynamic processes that lead to rhythmic behavior. Of such systems, anaerobic glucose metabolism in yeast is one of the most commonly studied. In a particular region of parameter space, all of the glycolytic intermediates show an oscillatory behavior with a variation in the frequency of oscillation observed across species. In the vicinity of the attractor that is responsible for these oscillations, the system never reaches steady state and hence this behavior cannot be readily analyzed by equilibrium, stoichiometric approaches such as metabolic flux balance analysis (Varma and Palsson 1994). We use this oscillatory system to demonstrate the capability of our approach to infer without constraining the equations governing a nonlinear dynamical metabolic system.

Our experiments involved placing the yeast glucose model (Figure 17.6A), in a numerical black box and then allowing our algorithm to conduct experiments on this black box. For our studies, we collect data by numerically integrating the differential equations in the black-box glycolysis model and adding noise. Initial states are constrained to a specified range, and the initial states for the test data are sampled over a larger volume in state-variable space to determine how well models can extrapolate and predict new behavior.



**Figure 17.6. Reaction networks for anaerobic metabolism in a yeast cell. Left:** The exact model includes membrane transport of glucose and pyruvate/acetaldehyde. Reactions in red involve ATP production/usage, and reactions in blue involve redox species production/usage. **Middle:** The impaired model does not produce either glycerol or ethanol. **Right:** The overspecified model has an additional sink for pyruvate/acetaldehyde ( $S_4$ ).

Our goal is to find the exact differential equations of the unknown system algorithmically. More specifically, we are interested in modeling metabolic networks as a dynamical system – a set of ordinary differential equations. In a system of  $N$  state-variables that we observe experimentally, (*e.g.*, extracellular concentrations of glucose ( $S_1$ ) or NADH ( $N_2$ ) over time), we must identify  $N$  (possibly nonlinear) differential equations. Synthesizing the mathematical models of a dynamical system is the most computationally intensive task in our procedure. We first smooth and then differentiate the observed time-series data to produce its derivatives. We then search for the differential equations that reproduce each numerically estimated derivative.

We calculate the numerical time derivative of each variable in the dataset so that we can measure error of each candidate differential equation explicitly without numerical integration, using the measurements of other variables in lieu of their yet-unknown equations (Bongard and Lipson 2007). The time to find a set of equations thus grows nearly linearly with the number of equations. However, the number of experiments and the time to find each differential equation depend primarily on the complexity of each equation's expression. The simplest equation,  $S_5$  (external pyruvate/ acetaldehyde), required approximately one minute for  $\sim 3 \times 10^6$  evaluations, and  $\sim 1$  model/experiment/evolution cycle. In contrast, the time to regress one of the most complex differential equations in the glycolysis model,  $A_3$  (ATP), was approximately 1-2 hours, and involved  $\sim 4 \times 10^{11}$  point evaluations on four workstations (eight 2.4 GHz cores), representing  $\sim 200$  model/experiment evolution cycles. Figure 17.7 shows correlation plots of the top-ranked individual during regression for each differential equation.

We conducted ten independent trials to collect data and model each equation. Figure 17.7 shows the runs that reached the highest performance on the training data (blue).



**Table 17.5. The differential equations describing glycolytic oscillation of the generating model (left pane) and the inferred model from the training data, which had 10% noise (right pane).**

<i>Original system</i>	<i>Automatically inferred system</i>
$\frac{dS_1}{dt} = 2.5 - \frac{100 \cdot A_3 S_1}{1 + 13.68 \cdot A_3^4}$	$\frac{dS_1}{dt} = 2.53 - \frac{98.79 \cdot A_3 S_1}{1 + 12.66 \cdot A_3^4}$
$\frac{dS_2}{dt} = \frac{200 \cdot A_3 S_1}{1 + 13.68 \cdot A_3^4} - 6 \cdot S_2 - 6 \cdot S_2 N_2$	$\frac{dS_2}{dt} = \frac{200.23 \cdot A_3 S_1}{1 + 13.80 \cdot A_3^4} - 6.87 \cdot S_2 - 6.87 \cdot N_2 + 0.95$
$\frac{dS_3}{dt} = 6 \cdot S_2 - 6 \cdot N_2 S_2 - 64 \cdot S_3 + 16 \cdot A_3 S_3$	$\frac{dS_3}{dt} = 6.00 \cdot S_2 - 6.00 \cdot N_2 S_2 - 64.16 \cdot S_3 + 16.08 \cdot A_3 S_3$
$\frac{dS_4}{dt} = 64 \cdot S_3 - 16 \cdot A_3 S_3 - 13 \cdot S_4 - 100 \cdot N_2 S_4 + 13 \cdot S_5$	$\frac{dS_4}{dt} = 64.04 \cdot S_3 - 16.03 \cdot A_3 S_3 - 13.03 \cdot S_4 - 100.11 \cdot N_2 S_4 + 13.21 \cdot S_5$
$\frac{dN_2}{dt} = 6 \cdot S_2 - 18 \cdot N_2 S_2 - 100 \cdot N_2 S_4$	$\frac{dN_2}{dt} = -0.055 + 5.99 \cdot S_2 - 17.94 \cdot N_2 S_2 - 98.82 \cdot N_2 S_4$
$\frac{dA_3}{dt} = -1.28 \cdot A_3 - \frac{200 \cdot A_3 S_1}{1 + 13.68 \cdot A_3^4} + 128 \cdot S_3 + 32 \cdot A_3 S_3$	$\frac{dA_3}{dt} = -1.12 \cdot A_3 - \frac{192.24 \cdot A_3 S_1}{1 + 12.50 \cdot A_3^4} + 124.92 \cdot S_3 + 31.69 \cdot A_3 S_3$
$\frac{dS_5}{dt} = 1.3 \cdot S_4 - 3.1 \cdot S_5$	$\frac{dS_5}{dt} = 1.23 \cdot S_4 - 2.91 \cdot S_5$

Additionally, we measured performance on the test data (shown in red), as described earlier. Variables  $S_3$  (1,3-bisphosphoglycerate)  $S_4$  (cytosolic pyruvate and acetaldehyde pool),  $N_2$ , and  $S_5$  were the fastest equations to infer, and their performance curves gradually converge monotonically during regression. Equations for  $S_1$ ,  $S_2$  (Glyceraldehydes-3-phosphate and dihydroxyacetone phosphate pool) and  $A_3$ , which have the most nonlinear structure, show performance that is more rugged. Dips in the training data performance indicate that data from a new experiment revealed dynamics that were not in the current data set (or perhaps underemphasized). Such dips tend to precede large improvements in performance. The equations with the best fits to the *training data* in ten trials are shown in Table 17.5.

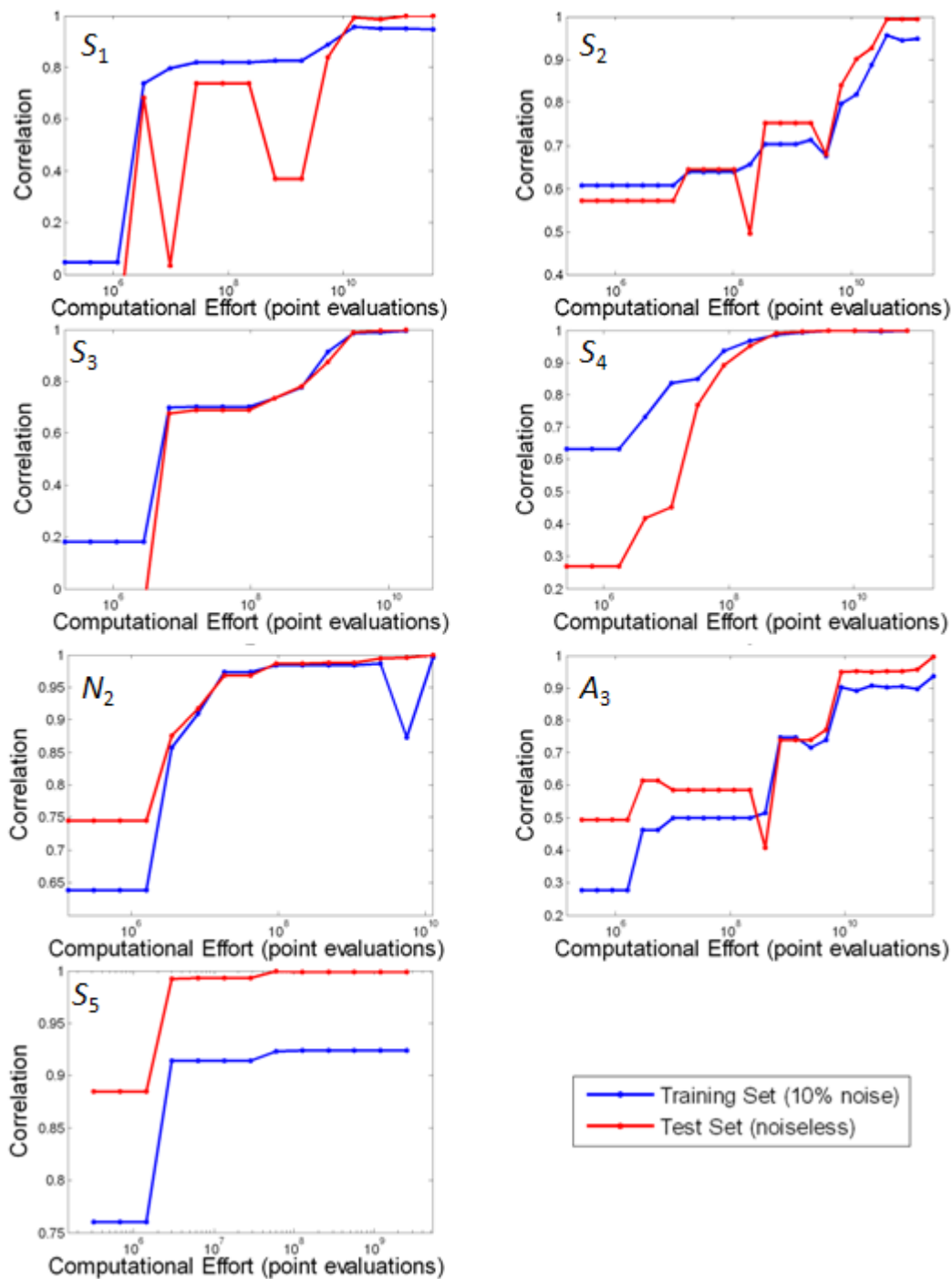
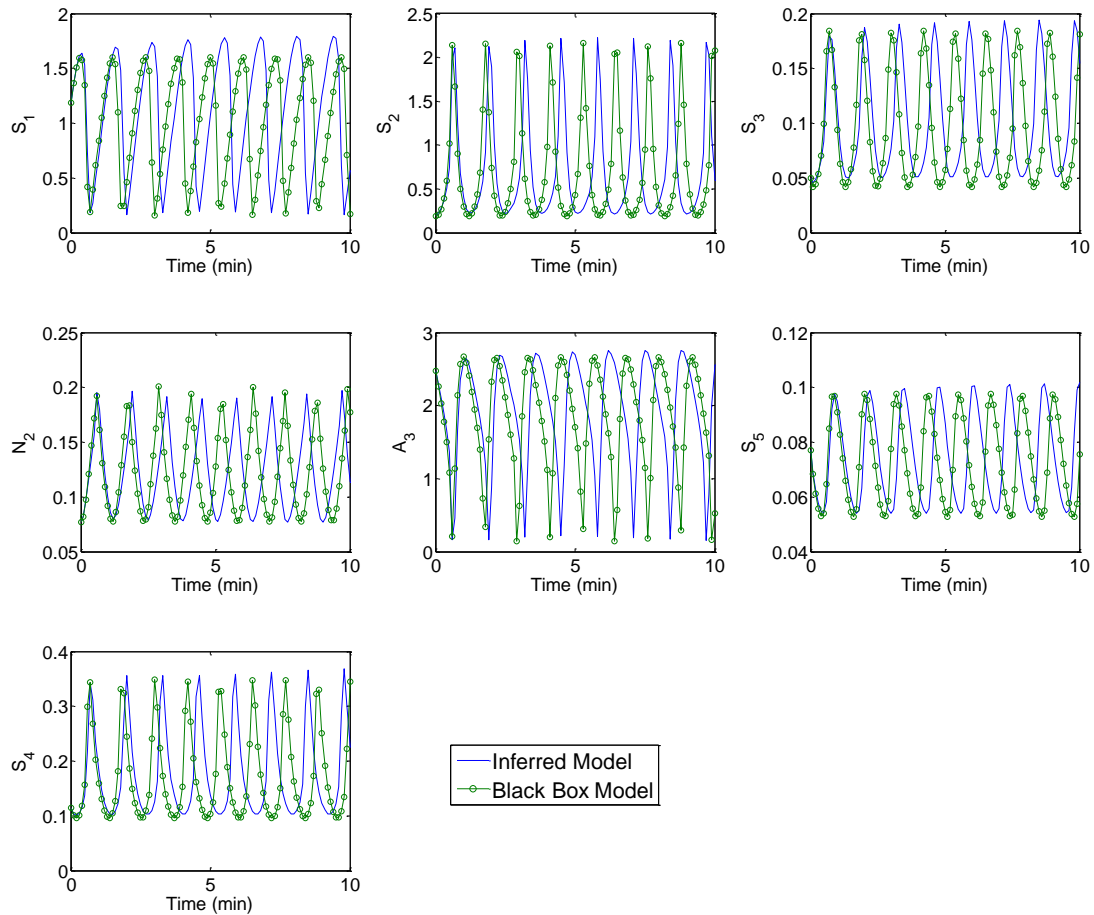


Figure 17.7. The fit to the data of the highest ranked solution during regression for each glycolysis variable. The blue series show the correlation coefficient to the training data, and the red to the test data. The training data contain 10% noise while the test data have none. The test data contain a larger range of allowed state variables (*i.e.*, sampled with weaker constraints) to measure whether the model can extrapolate and predict new behavior.

The automatically inferred equations are nearly identical to the black-box generating numerical model. Some slight differences remain: most notably, the parameters are inexact, which results in a slight mass imbalance, and one nonlinear term is approximated by a linear term in the  $S_2$  equation.

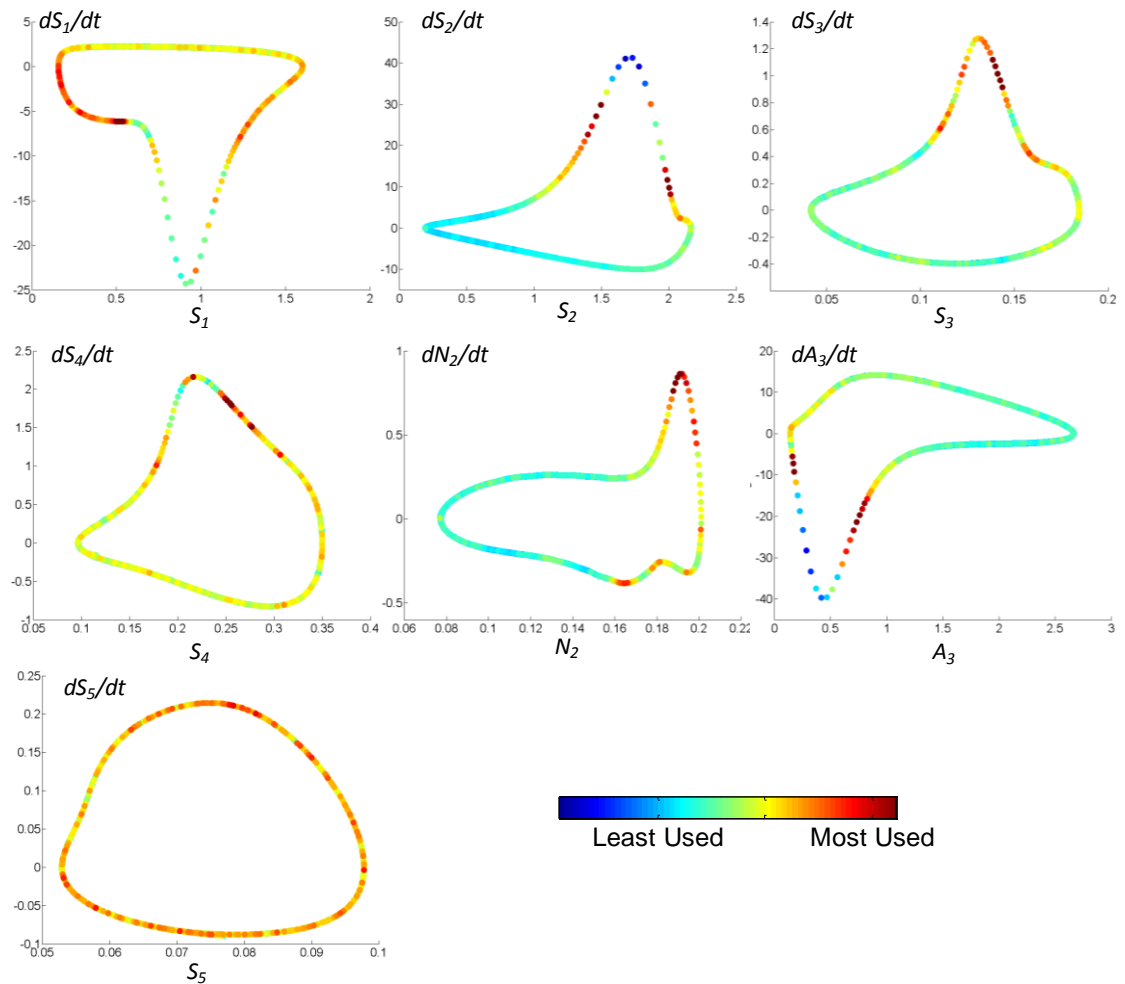
Integrating the inferred model (shown in Figure 17.8), however, shows the same behavior as the original system. Since symbolic regression does not have any inbuilt "chemical logic," it is unable to recognize and constrain reaction rate expressions that appear in multiple ODEs. The ramifications of this are twofold: first, the inferred model incurs small mass imbalances within the system that manifest themselves as a carbon loss or a source term in the energetic pools (ATP and NADH); second, the inferred model compensates for the imbalances by adding compensatory terms. These terms manifest themselves in the equations for  $S_2$  and  $N_2$ . Mass-balance logic can be built into future implementations, albeit with some performance costs (see "Discussion").

Specifically, the  $S_2$  equation approximates the  $N_2 \cdot S_2$  term and adds a constant term (0.9467) to the ODE. The  $N_2 \cdot S_2$  term comes from the balance of  $v_2$  and  $v_6$ , where  $v_2$  is the conversion of  $S_2$  to  $S_3$ , and  $v_6$  is the loss of  $S_2$  to glycerol production. Both the  $v_6$  and  $v_2$  fluxes are NAD dependent, which gives rise to the  $N_2 \cdot S_2$  term through a simple application of mass action kinetics. The decoupled  $N_2 \cdot S_2$  dependence is now represented it as a linear combination of  $(N_2 + S_2)$ . For the  $N_2$  equation, the combined action of  $v_2$  and  $v_6$  are properly inferred. However, there is (once again) a small constant term (-0.0549) in the  $N_2$  ODE that compensates for the fact that the NAD pool is not being strictly conserved.



**Figure 17.8.** The exact black box model and inferred model integrated over time. The inferred model shown in Table 17.5 differs from the exact model by a slight mass imbalance. Integrated over 10 minutes, the inferred model captures the same behavior. While small differences in derivative values tend to accumulate during integration, the inferred model captures the integrated behavior remarkably well. The inferred model predicts early behavior accurately and exhibits the same qualitative dynamics later in time, differing only slightly in the phase.

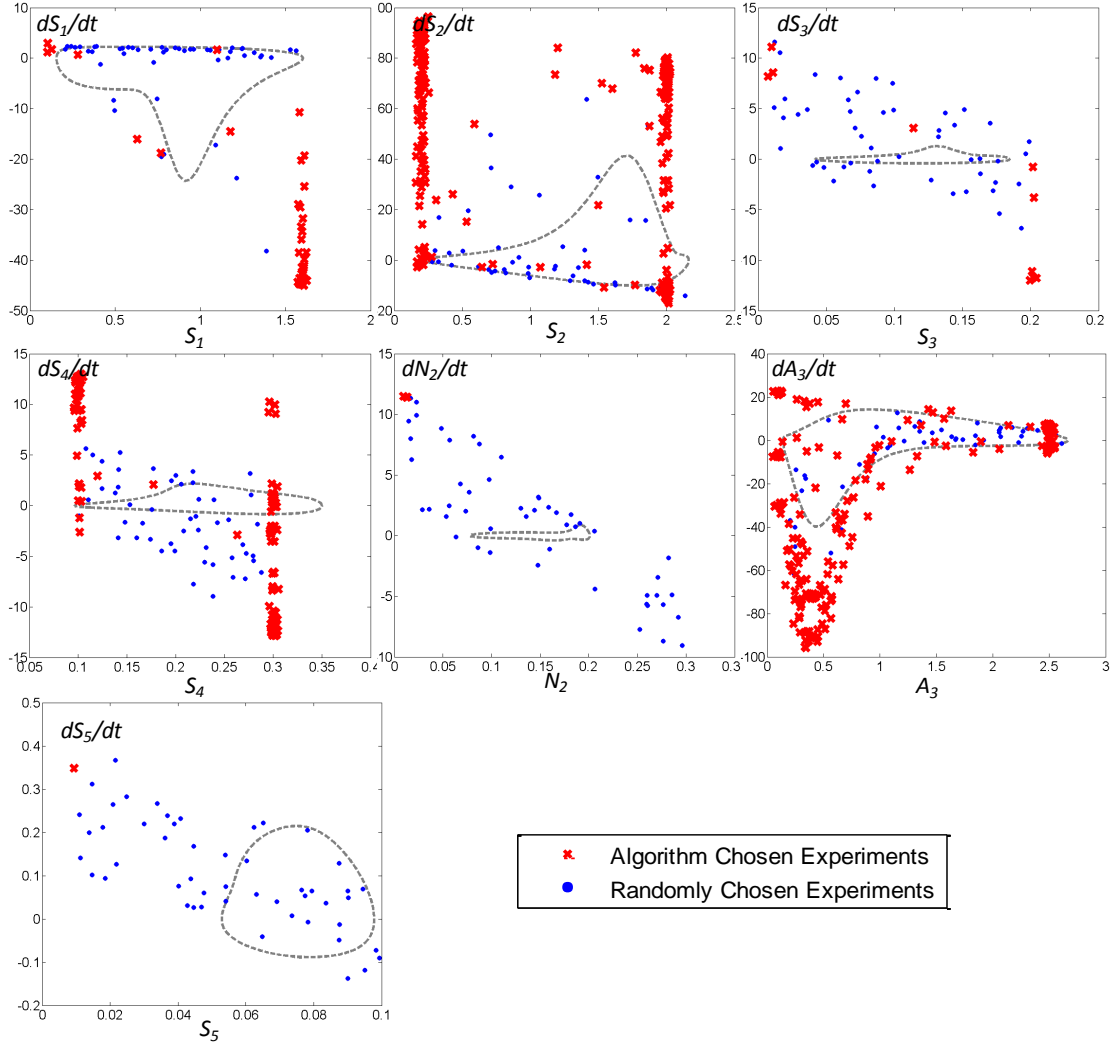
When looking at the experimental tests the algorithm chose during regression, it is not immediately obvious what data and initial conditions are most informative in a seven-dimensional domain. However, we can pick out some basic empirical trends. Figure 17.9 shows the most differentiating data points among the population of equations within a single time-series. The left side of the figure provides a phase-space representation of each variable and its time-derivative. The points in these trajectories



**Figure 17.9** The glycolysis system near the stable limit cycle in the course of a single experiment, with colors representing frequency with which the fitness predictor examines each point within a single time-series.

are color coded by the frequency of their use calculating comparing equations (via the fitness prediction). In a single time-series, the importance (frequency of references by the fitness predictors) of a given point is not necessarily those system states with high derivative magnitudes. Instead, heavy importance tends to lie near inflections around the limit cycle for most variables.

The right half of Figure 17.10 shows the range of initial conditions (red) chosen by the estimation-exploration algorithm as it suggests new experiments for each iteration in



**Figure 17.10.** The initial condition experiments (red) chosen by the algorithm to differentiate solutions in comparison to a random distribution of initial conditions (blue). The algorithm tends to focus on nonlinear states away from the limit cycle (dashed black line) within the experimental constraints imposed upon the estimation-exploration algorithm.

the series. The blue points show the range of derivative values for randomly chosen states. The dashed line shows the limit cycle for each variable.

With the exception of  $A_3$ , the EEA is preferentially choosing new experiment initial conditions near extremities of the allowed range of each variable, away from the limit

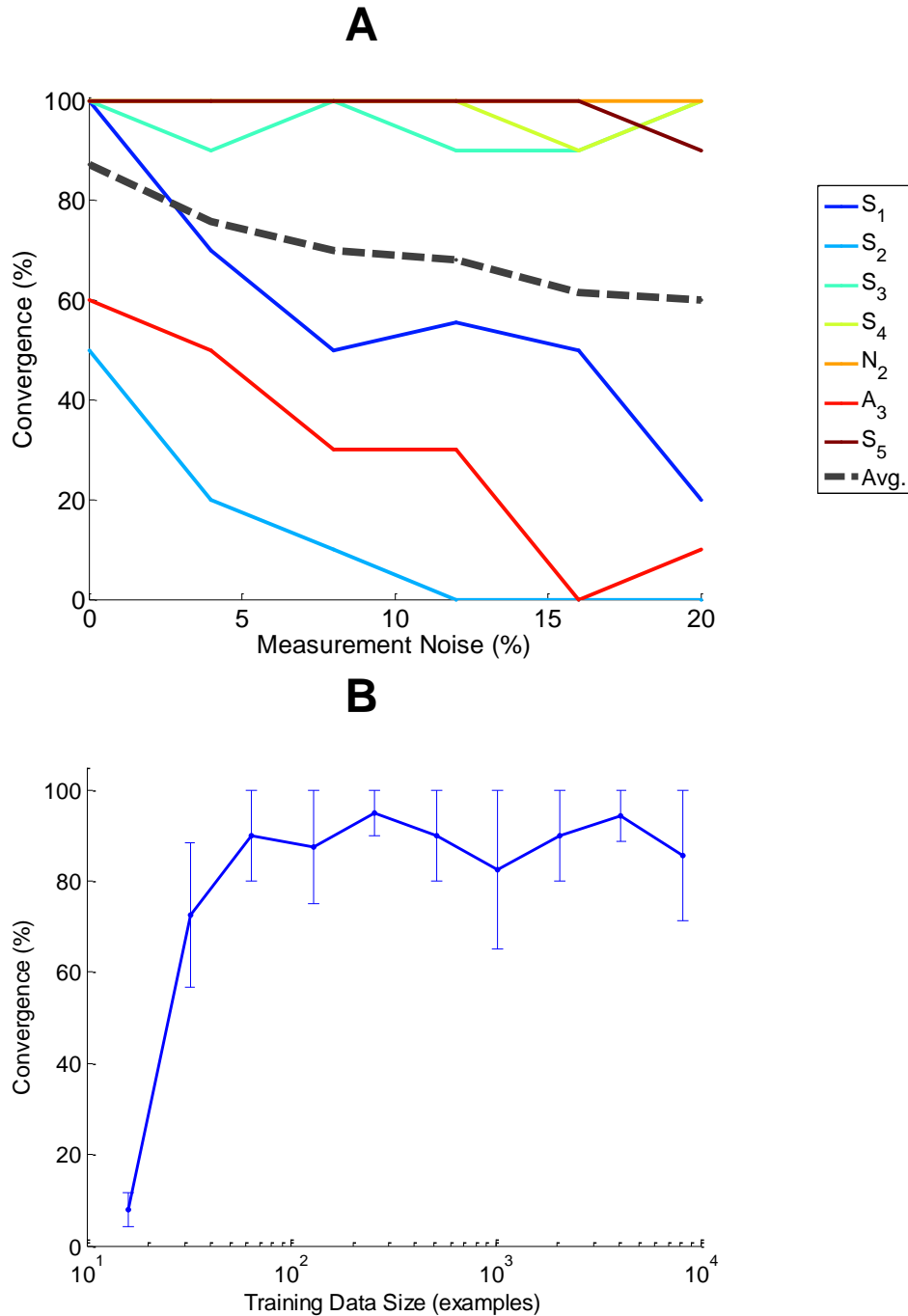
cycle. These initial conditions are more likely to amplify nonlinear features of the system, which is also consistent with the observed behavior of the fitness predictor. Therefore, the maximum disagreement criterion for new experiments may in effect reveal information about the nonlinear terms, which appear to be the most used data points for synthesizing models within single trajectories. Initial conditions and measurements on the limit cycle provide much less information than ones that lie outside the limit cycle for which the system must descend into the limit.

The amount of noise in the system affects the frequency of finding the exact differential equation for each state-variable differently. Figure 17.11 shows the rate of convergence (success rate) for each equation within one hour of regression. The most complicated differential equations ( $S_1$ ,  $S_2$ , and  $A_3$ ) are also the most sensitive to noise. We have found that noise obscures subtle features in these equations, resulting in partial regression of the exact equations. For example, in the solution for  $S_2$  in Table 17.5, the  $v_1$  reaction term is found exactly, but the  $v_4+v_6$  reactions are approximated.

### ***Sequence of Solutions***

Since symbolic regression begins with randomly generated solutions (differential equations), it is interesting to observe the evolutionary path these solutions take toward the final model. Table 17.6 shows one evolutionary sequence for the  $S_1$  variable.

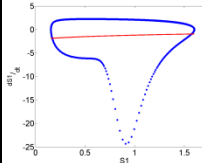
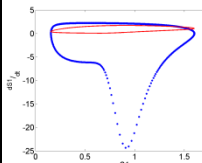
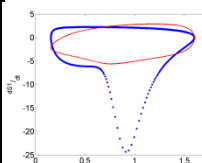
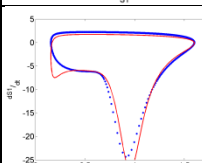
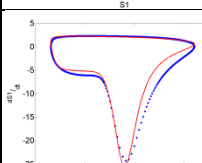
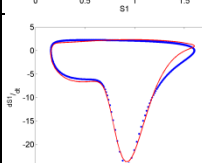
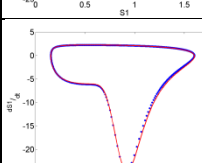
The solutions tend to grow gradually in complexity from the initially random solutions. The fit to the data improves incrementally. Finally, a solution that contains most of the exact model emerges, and the solution prunes down as it fits the last remaining features.



**Figure 17.11. (A) The rate of successful inference of the exact differential equation for each state-variable versus the observation noise in the system after one hour of regression. The convergence rate is calculated from ten independent trials on each equation at each noise level. (B) The rate of successful inference of the exact differential equation for all variables versus the total amount of data given to the system. The error bars indicate the standard deviation in convergence among the seven variables.**



**Table 17.6. Seven snapshots of the best solution during regression of  $S_1$ . The solution is plotted in red and the systems limit cycle is shown in blue.**

Generation	Fit to Limit Cycle	Current Best Model
2		$\frac{dS_1}{dt} = \frac{-2.5028}{S_2 + S_1 + 1}$
190		$\frac{dS_1}{dt} = \frac{A_2}{S_2 + 1.4659}$
2,605		$\frac{dS_1}{dt} = \frac{5.9310(A_2 - 1.6763)}{S_2 - S_1 + A_2 + 0.1587}$
316,029		$\frac{dS_1}{dt} = \frac{S_4 + 2S_1 + 2N_2 - 2A_2(A_2 - 0.9450) - 0.9950}{A_2(0.9450 - A_2) - 0.2948}$
407,083		$\frac{dS_1}{dt} = 2S_2 + \frac{2S_1}{(0.7557 - A_2) \cdot A_2 - 0.2046} + \frac{2N_2}{A_2} + 2.1192$
2,835,858		$\frac{dS_1}{dt} = ((2.1623 - A_2) \cdot ((-2.1623 - A_2)^2 - 1.079)) \cdot S_1 + 1.82$
4,444,185		$\frac{dS_1}{dt} = 2.5308 - \frac{42.3825S_1}{5.4326A_2^2 + \frac{0.4290}{A_2}}$

### *Predictive Accuracy Compared to Other Methods*

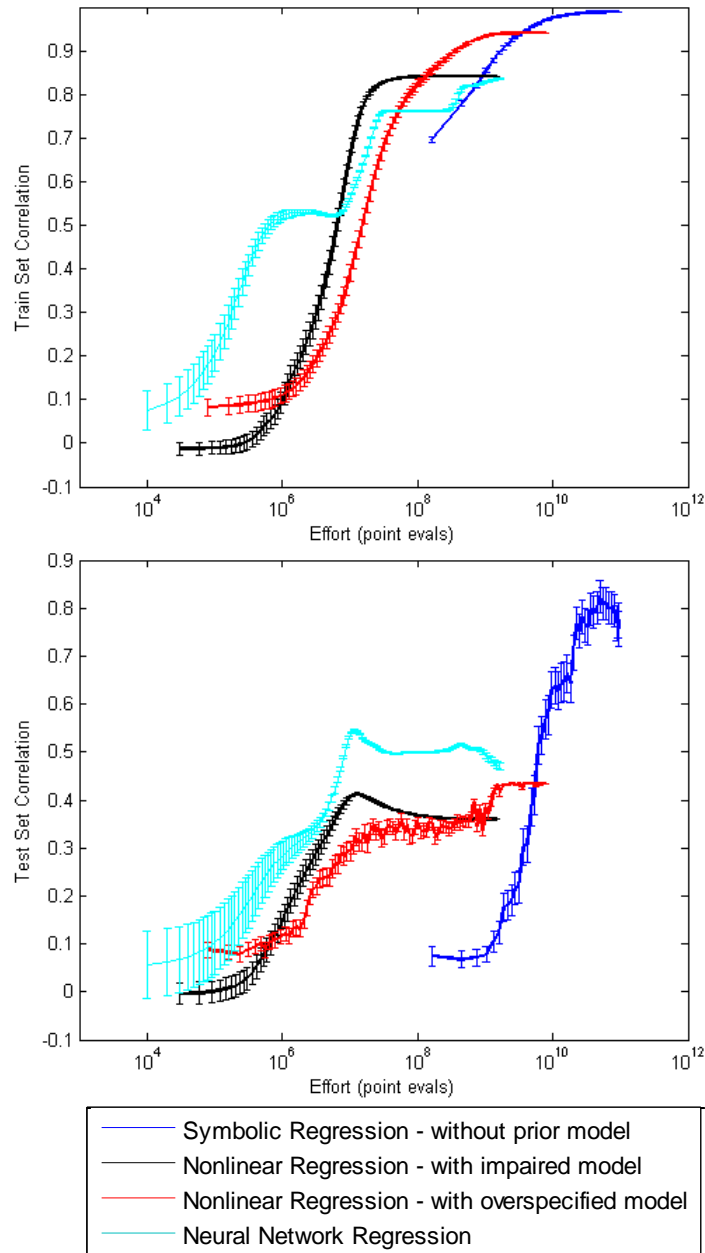
Searching the space of symbolic differential equations is unique in that it does not require prior information about the system or a prior model. Hence it is difficult to

compare this model inference process with the process of historical development of an existing metabolic model. To demonstrate the capabilities of this approach to specify a compact representation of a model, as might be needed for prediction and control of metabolism in a bioreactor, we have compared our approach with two relevant methods: nonlinear regression and neural-network regression.

Neural networks are recognized as being useful for predicting a time series when the underlying mechanism is unknown or is too complex to be easily represented, or noisy data limits the analysis (Crampin, Schnell et al. 2004). Such numerical models are less amenable to human interpretation, but can model and predict data similar to data used for training. In the neural network regression, we use a 1024-neuron hidden layer network mapping the seven state-variables to their seven respective time-derivatives. The output layer consists of linear perceptrons. We use standard back-propagation to train the network and bias-node weights.

In nonlinear regression, a preexisting mathematical model is chosen to be fitted to the data. The selected preexisting model is assumed to closely relate to the actual, underlying model of the system but may have a slightly different structure – it may be missing key terms, or have unnecessary terms, or incorrect terms. Regressing data to the wrong model may provide a low-error fit, but with parameter values that are in fact quite different from the actual ones because of the adjustments required to fit the wrong model to the data.

For the nonlinear regression comparison, we chose two slightly different preexisting glycolysis models, also shown in Figure 17.6 – one that is a simplification of the exact model (the “impaired” model) and one that is more complex in that it replaces some dynamics with an additional sink term (the “overspecified” model).



**Figure 17.12. Performance comparison between symbolic regression, nonlinear regression, and neural network regression. Results are averaged over 100 trials – error bars represent the standard error. Training data performance (top pane) shows that all algorithms accurately explain the training data. The negative slope of the correlations when the results from the training regression are applied to the test data indicates varying degrees of overfitting. Note that symbolic regression uses more point evaluations in the same amount of running time because it is a parallel search, whereas nonlinear regression and neural network back-propagation use serial updates.**

**Table 17.7. The equations for  $S_4$  (pyruvate and acetaldehyde pool) for the exact, impaired, and overspecified models shown in Figure 17.6. The exact values for the parameters are  $k_3 = 16$ ,  $k_4 = 100$ , and  $A_sP/V = 13$ .**

Model name	Differential equation	Regressed parameters
Exact model	$\frac{dS_4}{dt} = k_3 S_3 A_2 - k_4 S_4 N_2 - \left( \frac{A_s P}{V} \right) (S_4 - S_5)$	$k_3 = 16.03, 16.01$ $k_4 = 100.11$ $\frac{A_s P}{V} = 13.21, 13.03$
Impaired model	$\frac{dS_4}{dt} = k_3 S_3 A_2 - \left( \frac{A_s P}{V} \right) S_4$	$k_3 = 13.76635$ $\frac{A_s P}{V} = 21.2331,$
Overspecified model	$\frac{dS_4}{dt} = k_3 S_3 A_2 - k_4 S_4 N_2 - \left( \frac{A_s P}{V} \right) (S_4 - S_5) - \frac{k_{sink} S_4}{1 + \left( \frac{A_3}{K_{IATP}} \right)^3}$	$k_3 = 15.8508$ $k_4 = 94.812$ $\frac{A_s P}{V} = 12.0785$ $k_{sink} = 0.411579$ $k_{ATP} = 0.5264$

The impaired model is produced by eliminating the glycerol ( $v_6$ ) and ethanol ( $v_4$ ) production and having the NADH→NAD recycle occur with the production of lactate from pyruvate. This essentially converts the yeast model into that of a mammalian cell. The overspecified model is produced by relaxing the assumption of no carbon loss to other cellular synthetic processes in the yeast model (fatty acid biosynthesis, amino acid production, etc.). This is accomplished by adding a carbon sink term ( $v_{sink}$ ) to the pyruvate pool, whose rate is primarily controlled by the presence of ATP.

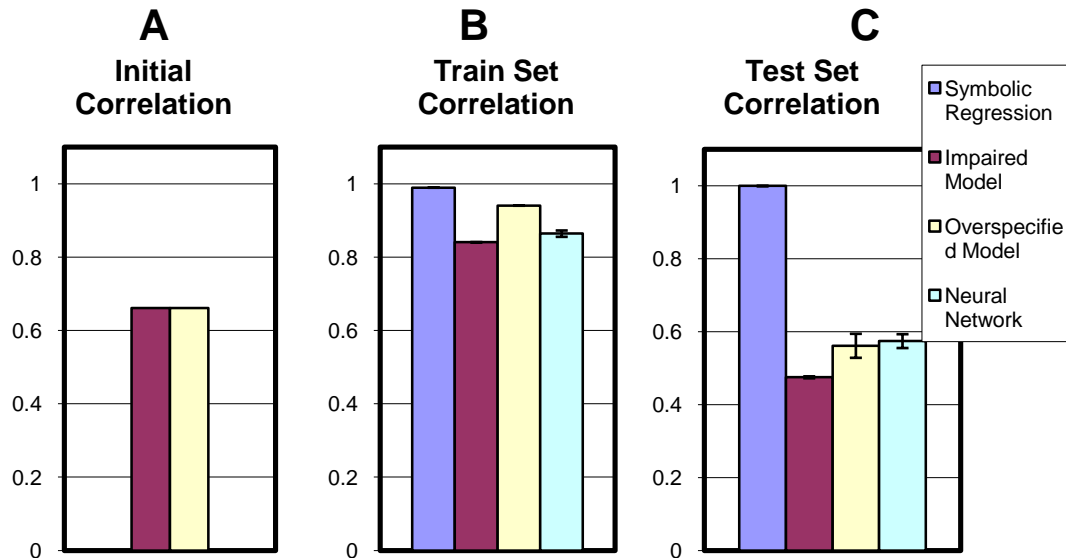
We tested performance of each method on modeling the time-derivative of  $S_4$ , the equation that differs the most between the impaired and overspecified nonlinear regression models. The symbolic regression algorithm must search for and fit the equation from scratch, whereas the nonlinear regression and neural network modeling

must tune parameters. The training data are static and were generated using the exact model – there were no algorithmically chosen experiments. As before, the test dataset has an upper-bound constraint that is twice that used for the training data set. Additionally, the training dataset again contains 10% random noise on every measurement. We stop regression after the solutions stop improving when evaluated on the test data set.

On average, all four algorithms model the training data equally well, but some do not generalize well when the regression results are applied to the broader test dataset that was not used for training (Figure 17.12). It is clear that nonlinear regression of both related models can explain a substantial amount of the  $S_4$  dynamics, particularly within the training data. However, extrapolation to the wider domain of the test data only reaches correlation of approximately one half. Similarly, the neural network accurately models the training data, but as shown by the early dip in correlation, significantly overfits before converging, possibly due to the added noise. Figure 17.13 summarizes the best average results of these algorithms (see “Materials and methods”).

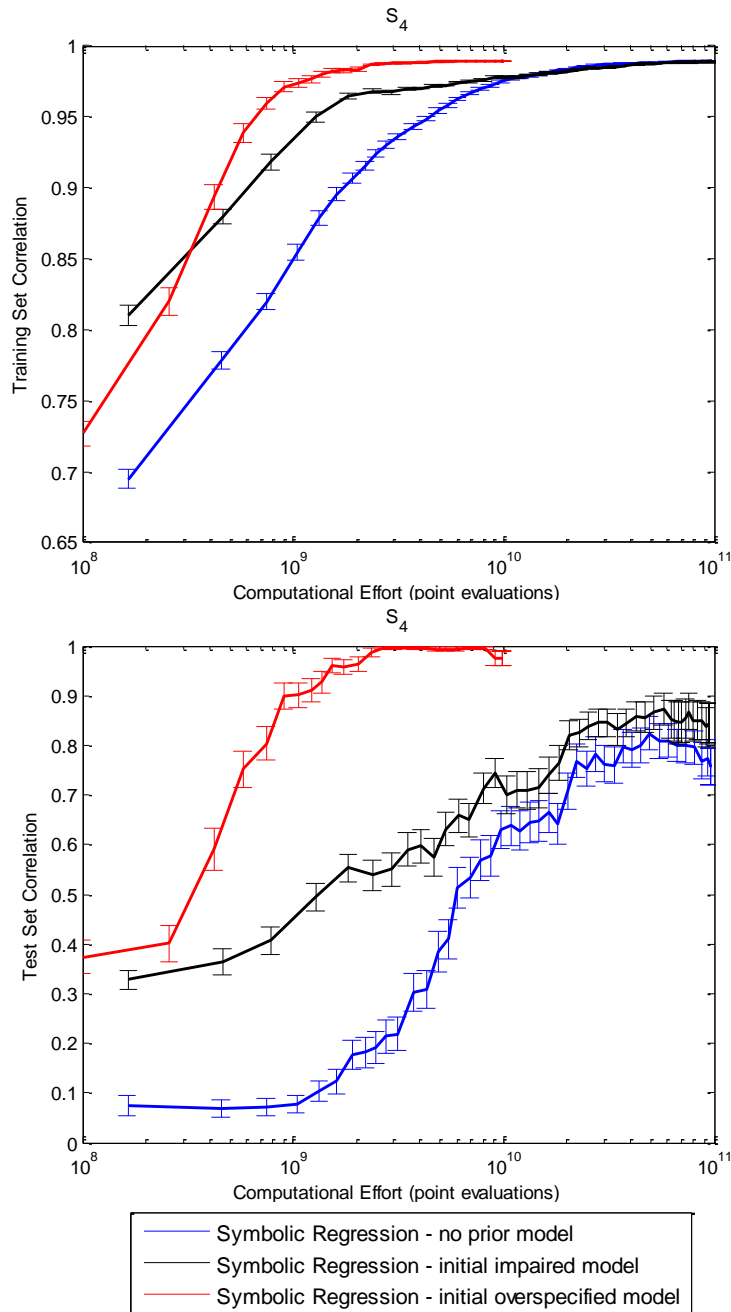
### ***Differentiating Hypothesized Models***

One useful application of the differential equation search is that it can also be used to adapt or improve existing hypothesized models. For example, the algorithm can modify a hypothesized model by altering its existing structure and terms, pruning unnecessary terms, or adding new terms to identify the exact intrinsic model. While methods for simplifying complex models already exist (Schmidt, Madsen et al. 2008), no previous method is able to compose new terms in a model or correct erroneous terms. Using automated data acquisition, the algorithm can also design experiments to differentiate hypothesized models and test their correctness where they disagree most.



**Figure 17.13.** Correlations of the various regressions averaged over 100 trials on equation  $S_4$  – error bars represent the standard error. (A) The correlations between the training data and each initial model before the model is regressed to the training data by the corresponding algorithm. Symbolic regression and neural network regression must model the system from scratch and initially have zero correlation. The impaired and overspecified models are close approximations to the exact model and therefore have positive correlations. (B) The mean correlation of the best solution from ten runs of each algorithm to the training data. The training data contain 10% random noise, which results in slight variances – most notably in the neural networks. The best solution from each algorithm correlates well to the training data with low standard error. (C) The mean correlation of each method to the test data. The assumed structures of the impaired and overspecified models limit their ability to model a wider phase domain. The neural network appears limited by noise in the system, but does achieve a higher correlation on average with the test set than do the impaired and overspecified models.

To use the algorithm to refine a given model or set of models, we modify its initialization by seeding the initial population with the chosen models. Effectively, this biases the algorithm to reuse the structure of the given model, but does not restrict the algorithm from making large alterations. We have tested the impact of seeding regression of the glycolysis  $S_4$  differential equation using the impaired and overspecified models.



**Figure 17.14. Performance comparison of symbolic regression when correcting a hypothesized model. Results are averaged over 100 trials – error bars represent the standard error. The blue curves represent the performance of the algorithm to the  $S_4$  equation without any prior model. For the other two pairs of curves, the symbolic regression algorithm was seeded with an incorrect hypothesized model (black = impaired, red = overspecified) and the algorithm had to modify the seeded model to fit the original training data. The graph shows the performance for both the training data used for the regression (top pane), and the test data (bottom pane) used to evaluate the training regression.**

There are two possible impacts that seeding a hypothesized model can have on symbolic regression. In the best case, the seeded model may be very close and the algorithm only needs to make minor adjustments to converge to the intrinsic model. Alternatively, the seeded model may be absurd, in which case there is no benefit and the algorithm evolves from scratch. Figure 17.14 exemplifies these two behaviors for the  $S_4$  differential equation.

In the overspecified case, the seeding has improved by a factor of ten both the speed to regress and the reliability in comparison to regressing from scratch. In effect, the algorithm has corrected the model by removing the sink term and fixing the differences in parameters. In the impaired model case, the effect is much less dramatic. The algorithm must construct the two missing terms. However, the seeding improves the reliability of convergence on average. There is also the possibility that the initial seeding might trap the algorithm in a local error minimum from which the present algorithm cannot escape. Similar problems have been encountered and addressed in conventional nonlinear regression schemes by, for example, choosing random data points at some distance from the regressed solution to ensure that it is in fact a global rather than local minimum. Similar approaches could be taken with our method, although it is unclear whether this would provide any advantage to regression from scratch.

## **Conclusions**

We have proposed a method for building ODE models of metabolic networks automatically from noisy experimental data. The modeling process explores the space of symbolic differential equations using symbolic regression for building mathematical equations and an estimation-exploration algorithm for designing new *in silico* or even wet-lab experiments to test and refine candidate models.



We demonstrated identifying the differential equations of the largest automatically inferred system to date, from noisy experimental data of a biological system. We showed *in silico*, with simulated measurement noise, regression of glycolytic oscillations in yeast. The glycolysis system studied exhibits a stable seven-dimensional limit cycle and contains subtle nonlinear terms that have taken rigorous investigation to unravel (Higgins 1964; Goldbeter and Lefever 1972; Richter, BETZ et al. 1975; Termonia and Ross 1981; Smolen 1995; Goldbeter 1996; Wolf and Heinrich 2000; Ruoff, Christensen et al. 2003).

We also compared the nonlinear regression of two approximate glycolysis models with symbolic regression and with neural network regression. While each algorithm modeled the training data well, searching for the exact differential equation generalized to the withheld test data most reliably and accurately (on a wider range of data that was not used for regression) without overfitting.

Finally, we showed that seeding symbolic regression with a human-hypothesized model or closely related model can significantly improve the ability and speed to find the exact model for the unknown metabolic network. In contrast to other techniques for model reduction, this approach can also expand a nonlinear model to include features not present in the hypothesized or baseline model, and, most importantly, design experiments that can test hypotheses and correct subtle differences using experimental data to identify how a particular metabolic system differs from one described by an established model – how genetic or environmental influences can affect the mathematics that describes the metabolic behavior of a system.

**Summary**

A key challenge in developing a dynamical model based on experimental data is determining what mathematical building blocks are necessary to explain the system's dynamics. Any mathematical model can be reduced down to various combinations of simpler building blocks, such as monomials or trigonometric terms, which greatly simplify both our conceptual understanding and the search space of the system model. Here, I propose a method using Functional Data Analysis (FDA) to discriminate between random expressions and meaningful building blocks which could be useful in building a full system model. Detecting individual building blocks is difficult because they can be coupled with other nonlinear terms. Functional linear regression however provides an elegant means for dealing with these terms so that we can evaluate the merits of the building block itself. I experimented with motion-captured data of an insect wing during hovering flight. The method distinguished a small number of building blocks from several hundred possible building blocks generated by an equation search algorithm. The building blocks distilled using FDA are amenable to human interpretation, providing hints to the physical processes occurring during insect flight.

**Introduction**

Building blocks are simple expressions which comprise a more complicated mathematical model. More precisely, if we think of an equation as a binary parse tree of mathematical operations (Figure 8.2), the set of building blocks contains all subtrees (sub-expressions) of the tree.

There are two main benefits to being able to determine building blocks given only experimental data. Building blocks give us insight into the internal physics of the system without having to know the entire system model. For example, we may be able to determine the rate expression for an individual chemical reaction in a pool of reactions occurring in a metabolic network. This may be particularly useful in complex systems, such as where many reactions produce or consume a certain chemical, but modeling all reactions at once is intractable.

Secondly, if we had a method for calculating the merits of a building block, we could potentially search for building blocks automatically, gradually building up the full system model. This may be a method for scaling into modeling enormously complex systems that currently exceed our ability to unravel or understand.

Determining a useful building block can be considered to fall under the notorious “credit assignment” problem in machine learning. In short, the credit assignment problem is how to score or weight the importance of individual components of a model when only given entire systems. For example, what is the importance of the gears of a bicycle for riding quickly? This clearly depends on the other components of bicycle, such as the gear ratio, the wheels, etc.

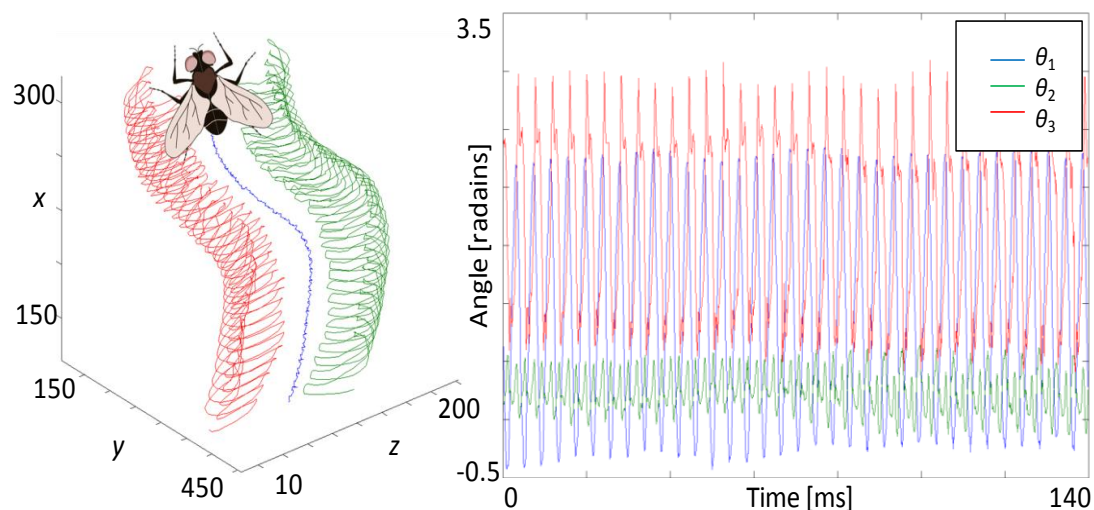
In the context of detecting building blocks, we are interested in asking how useful a component is for explaining the dynamics of the larger system. For example, a small term such as  $\omega(t)\cos(\theta(t))$  may capture an important nonlinear feature in the data, but on its own, cannot explain or fit all of the data very well. So, perhaps it is an essential term to the physical model of the system, such as an orientation dependent drag force. Could we detect this to be the case?

In the remaining sections I introduce the dataset I used for analysis, describe my method at recombining and registering the data, describe the building block detection method, and finally end with concluding remarks.

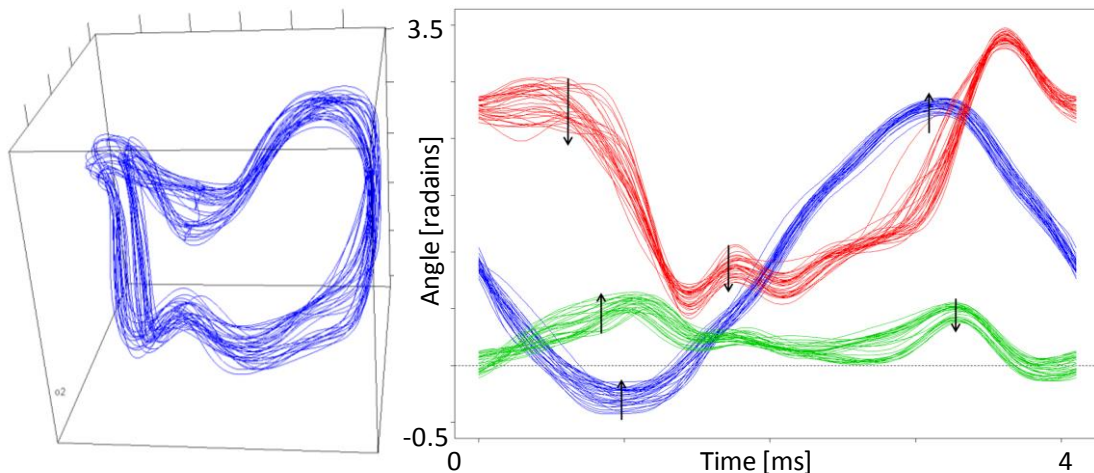
### The Insect Wing Data

The dataset I analyzed is from motion capturing of an insect wing during flight. The this dataset described in (Ristroph, Berman et al. 2009) was provided by Gordon Berman in the Cornell Dragonfly group (Figure 18.1). They captured video from three high-speed cameras at 1000 frames per second, and then did volume reconstruction of the wings by intersecting the volume projections of each camera.

The portions of the data I am looking at are the angular positions of the insect's right wing – namely yaw, pitch, and roll. The dataset covers approximately 34 consecutive periods of the insect flapping its wing.



**Figure 18.1. The tracked position of the fly (top pane) and the corresponding angles of the right wing (bottom pane). This data was recorded over approximately 34 flapping periods during 140 milliseconds of flight.**



**Figure 18.2.** The three dimensional plot (left pane) of the right wing angles ( $\theta_1$ ,  $\theta_2$ , and  $\theta_3$ ). There is slight variation among the periods but overall they line up neatly. After chopping up the periods, there is covariance between different peaks of each angle (right pane).

Plotted in 3D (Figure 18.2), we can see the periods overlap very closely. Described in the next section, I chopped up this time-series to compare the dynamics amongst the periods. Overall, the periods are very consistent, but there is some drift over time.

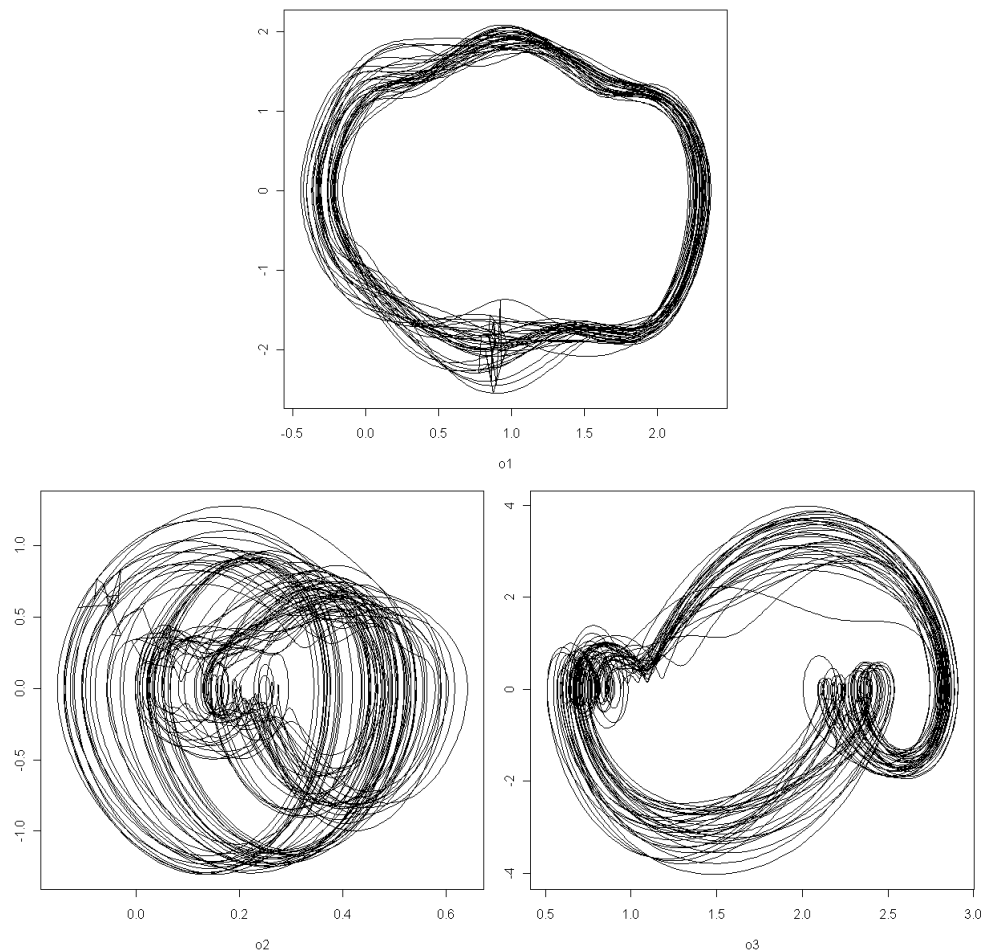
Looking at the phase plots, (plots of  $d\theta_i/dt$  vs.  $\theta_i$ ), we can see the limit cycles of each angle fairly clearly (Figure 18.3). The phase plot of the first angle  $\theta_1$  is nearly circular, suggesting it is a simple linear oscillator. The other phase plots however show more complex limit cycles, that likely have more complex building blocks explaining their dynamics. Building a functional linear model based on the period number explains much of the variation among the periods (Figure 18.4).

I also conducted PCA and multi-variate PDA on this data. The PCA analysis showed the first few principle components (PCs) focused on the magnitude of peaks of individual variables, further PCs were much harder to decipher. The PDA analysis was inconclusive, but suggested there could be some significant accelerations from drag forces (e.g. significant coefficients on velocity terms).

## Data Registration

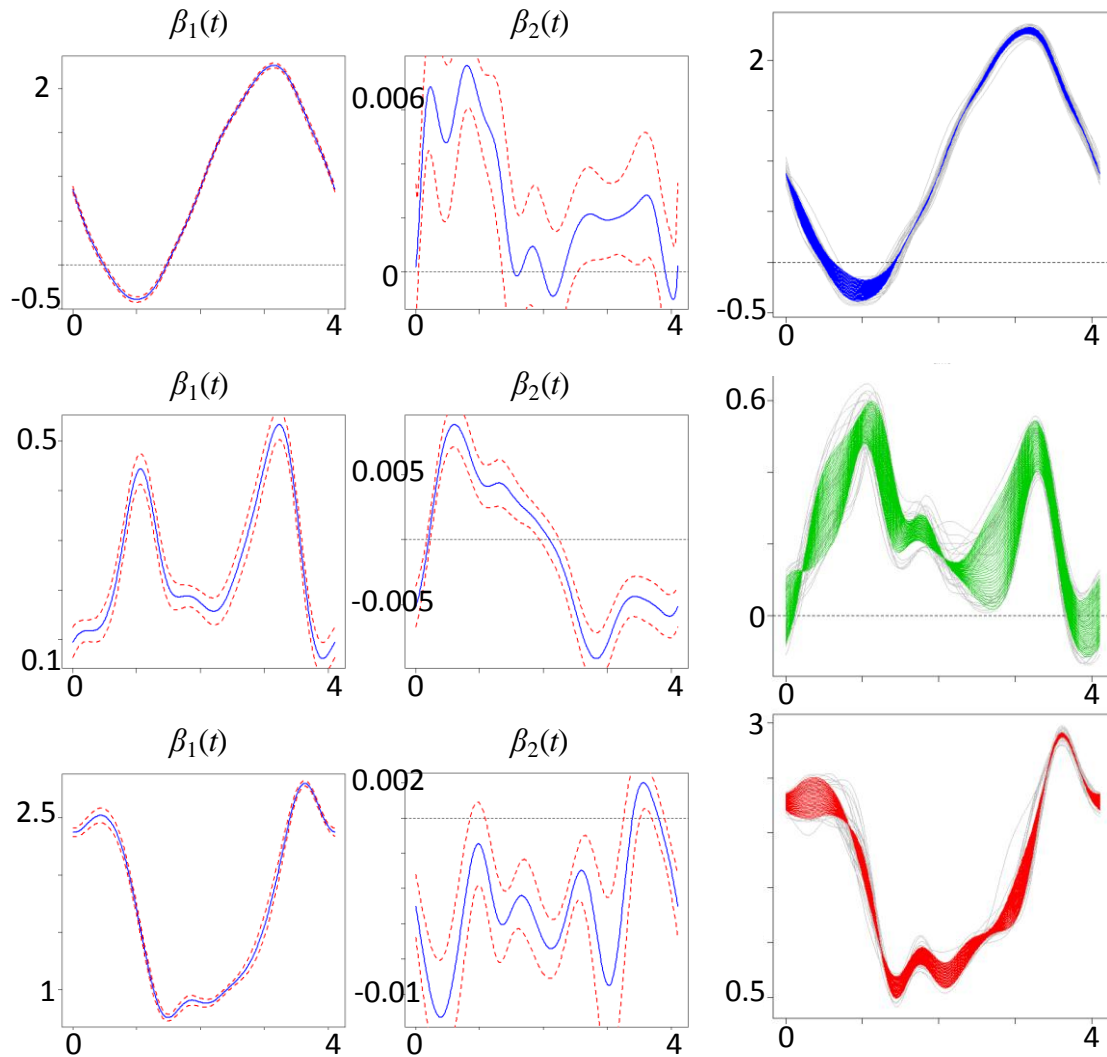
Data registration is the process of deforming the time axis of a dataset in order to make key, repeated features or events in the data overlap synchronously in time.

In registering this data, I first split the time series into periods for each wing flap. Choosing a mean period (mean time between peaks) resulted in poorly registered data. There appeared to be some drift in the period lengths over time.



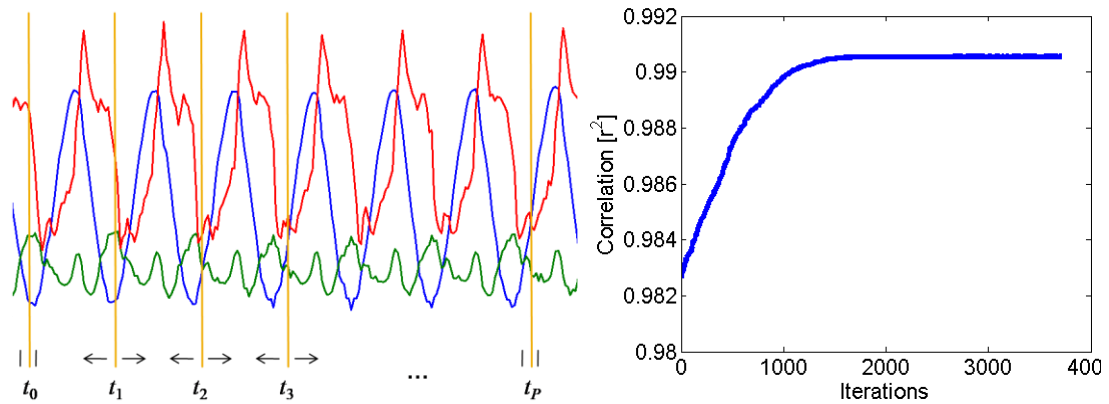
**Figure 18.3. Phase plots of the three angles of the right wing ( $d\theta_i/dt$  vs.  $\theta_i$ ), in order ( $\theta_1$ ,  $\theta_2$ , and  $\theta_3$ ). The first angle appears to be a simple harmonic oscillator, whereas the other two angles show more complex sub-cycles, likely containing higher order building blocks in their physical model.**

$$\theta(t) = \beta_1(t) + \beta_2(t) \text{Period}(t)$$



**Figure 18.4.** Functional linear models based on the period number explains much of the variation between periods. The linear coefficients (left), the fit and description of the drift (middle), and  $R^2$  scores (right).

To account for the drifting periods, I wrote a short program to register the slices of the periods. The method starts with an initial set of slices:  $S = (t_0, t_1, t_2, \dots, t_N)$ . Each slice of the data is rescaled to have the mean period length. The method then uses simulated annealing to stochastically vary the slice positions in order to greedily improve the correlation between all period slices (Figure 18.5).



**Figure 18.5.** The registration method slices the data into each periods, scaling length of each slice to have the same period (left pane). The method optimizes the positions of the slices in order to maximize the correlation among all the periods (right pane).

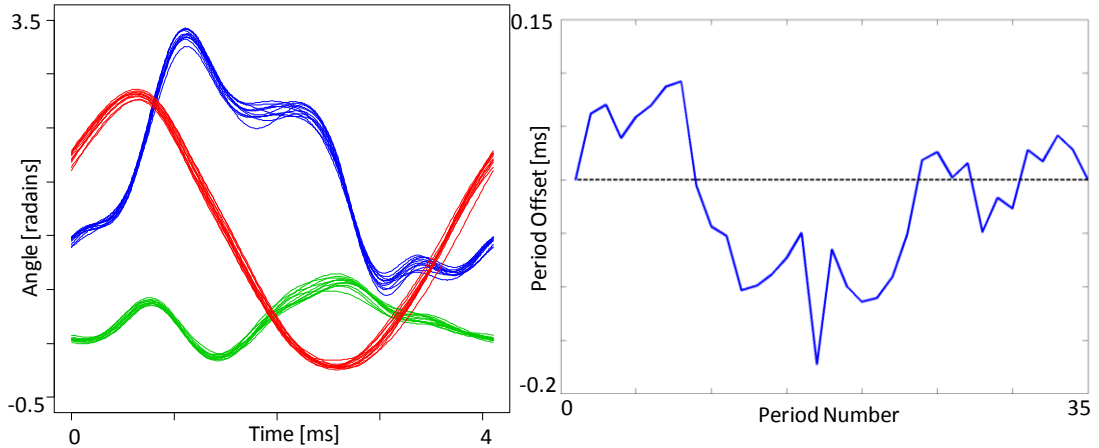
After the slice registration, I applied time warping registration on using the third angle (blue) which had the most distinct features (Figure 18.6). The time warp lines up smaller features within each period.

The positions of the slices (after optimizing for registration) reveal a slow drift in period length over time (Figure 18.6). Early on, the periods are longer than average. In the middle, the periods become shorter than average. The last periods settle near the average. This may correspond to the S-shaped flight positions of the fly shown in Figure 18.1. In which case, the period length (or frequency) of the wing flaps correlates with the lateral thrust, where the fly drifts right at low frequency, and left with high frequency in the right wing.

### **Building Block Results**

The major challenge to detecting a meaningful or useful building block is that the building block may be distorted by some nonlinear transformation in a higher order building block or it may be buried in variation from other terms. For example, say the exact equation for the angular acceleration of the second angle was the equation:





**Figure 18.6.** The registered data (left pane) and the shifts in periods after optimizing the slice positions (right pane). The periods over time drift slowly, correlating with the slight drift in the position of the fly in Figure 18.1.

$$D^2\theta_2 \approx 9.26 - 31.78 \cdot \sin(\theta_2) - 7.74 \cdot \cos(\theta_1 - 6.56) - 10.58 \cdot \cos(\theta_3 - 12.62) \cdot \cos(\theta_1 - 6.56)$$

Ideally, we could detect if a term such as  $\cos(\theta_1 - 6.564)$  is a building block ahead of time given various  $D^2\theta_2$ ,  $\theta_1$ ,  $\theta_2$ , and  $\theta_3$  data values. This term appears both linearly and multiplied with another term in the full exact model.

Functional linear regression provides an elegant way to abstract away the coupled and additive terms to the building block we wish to test. Consider fitting the following two-parameter functional linear model:

$$D^2\theta_2 = \beta_1(t) + \beta_2(t) \cdot \cos(\theta_1(t) - 6.56) + \varepsilon(t)$$

In this model, the basis expansions of the coefficients,  $\beta_1(t)$  and  $\beta_2(t)$ , can fill-in the blanks around the building block term to reproduce the exact signal: where  $\beta_1(t)$  assumes  $9.26 - 31.78 \cdot \sin(\theta_2)$  and  $\beta_2(t)$  mimics  $-7.74 - 10.58 \cdot \cos(\theta_3 - 12.62)$ . The building block in this linear model allows the  $\beta_1(t)$  and  $\beta_2(t)$  to be simpler, in that they do not need to reproduce the  $\cos(\theta_1(t) - 6.56)$  signal in addition to the other terms.

Therefore, we should be able to apply a more aggressive smoothing penalty to  $\beta_1(t)$  and  $\beta_2(t)$  in this model than a model with an incorrect building block, such as a random function:

$$D^2\theta_2 = \beta_1(t) + \beta_2(t) \cdot \text{random}(t) + \varepsilon(t)$$

In this extreme case,  $\beta_1(t)$  must reproduce  $D^2\theta_2$  on its own and  $\beta_2(t)$  must become zero in order to fit the data. Therefore, for the null case, we can force  $\beta_2(t)$  to equal zero and simply fit:

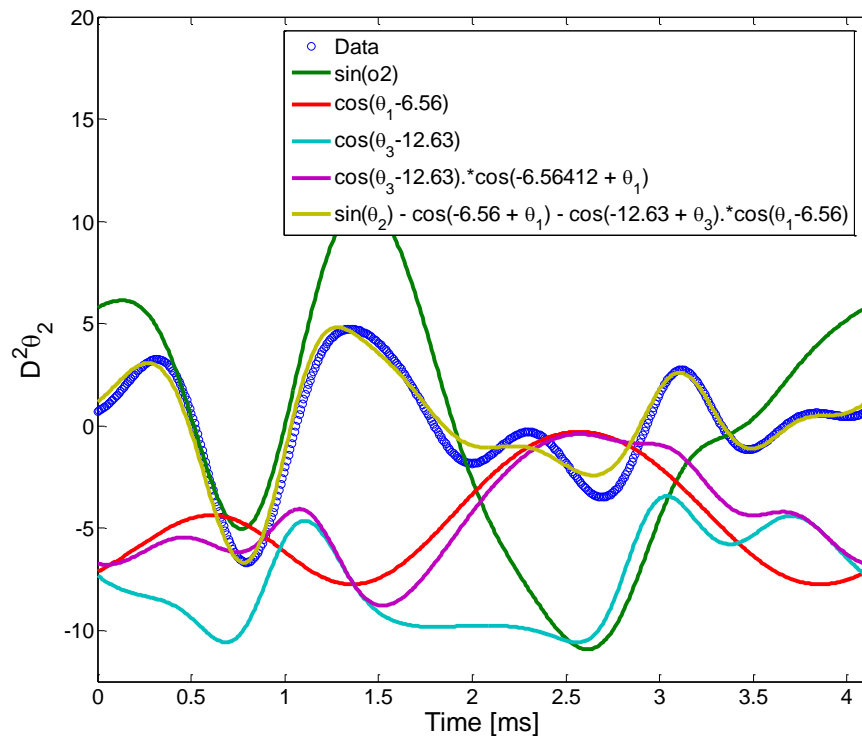
$$D^2\theta_2 = \beta_1(t) + \varepsilon(t)$$

Based on these observations, we can define a procedure for measuring how well a candidate building block helps explain the data. We sweep the  $\lambda$ -penalty for both the null model and the building block model and compare their cross-validation performance curves versus  $\lambda$ .

**Definition:** To be considered a successful building block, the cross-validation error should both reach a lower minimum than the null model, and remain at a low value for longer as the smoothing penalty increases. A poor building block will show a cross-validation performance very similar to the null model – either not improving the minimum error or not improving the tolerance to a higher smoothing penalty.

I examined the building blocks found for a model of the angular acceleration of the second angle,  $D^2\theta_2$ , which is shown in Figure 18.7. This equation was generated automatically using an algorithm, so we are not certain in advance the building blocks will show to be useful individually in the function linear models. In addition to detecting building blocks, we are also validating the components of this model.

$$f = 9.26 - 31.78 \cdot \sin(\theta_2) - 7.74 \cdot \cos(\theta_1 - 6.56) - 10.58 \cdot \cos(\theta_3 - 12.62) \cdot \cos(\theta_1 - 6.56)$$

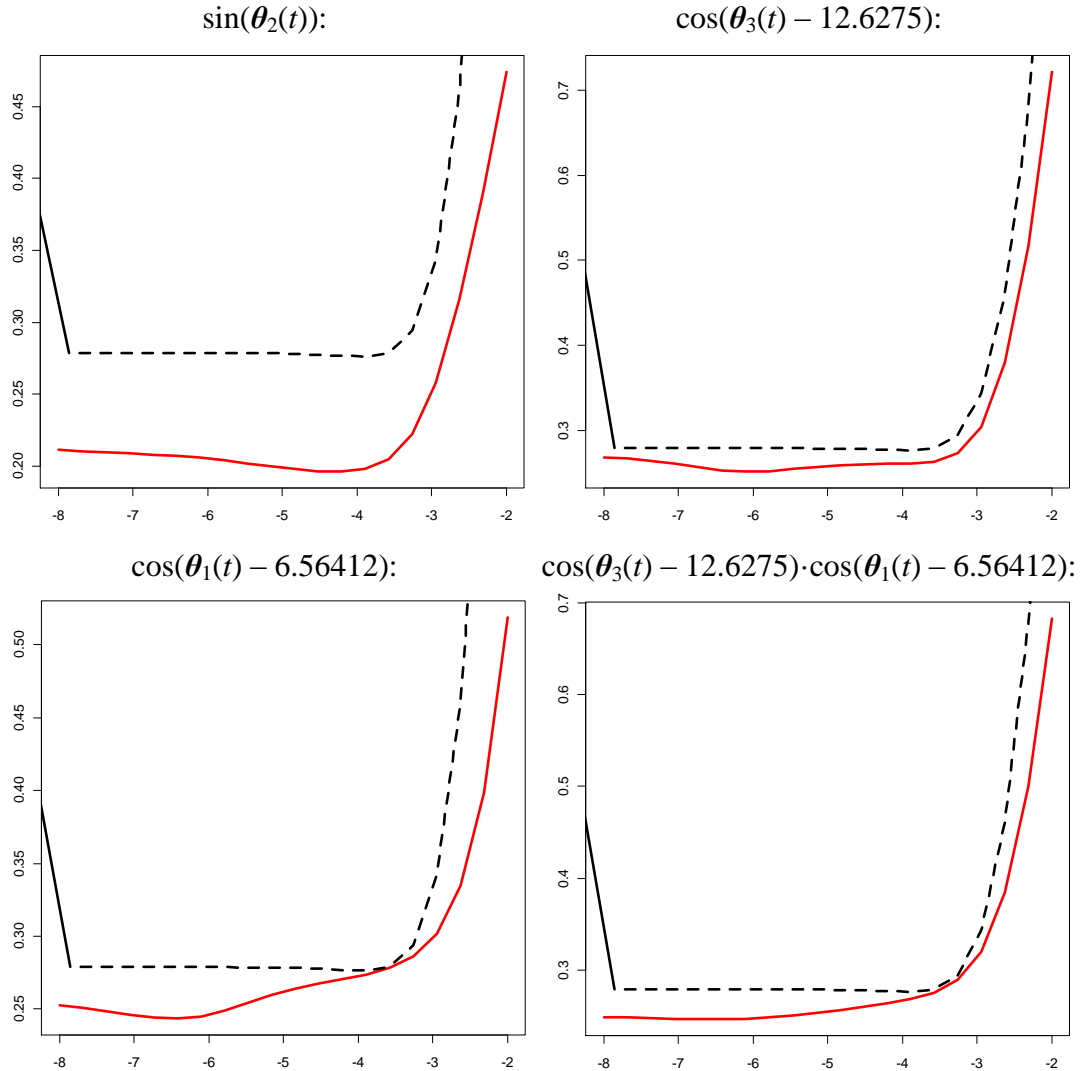


**Figure 18.7.** An equation modeling  $D^2\theta_2(t)$  and its individual building blocks. This equation was generated by an algorithm, so we are interested in testing whether its building blocks also show to be useful individually using the function linear model procedure.

The cross-validation scores of the four major building blocks (solid red) strictly dominate the scores of the null model (dashed black), reaching both lower minimum error, and spiking in error at a later  $\lambda$ -penalty. (see Figure 18.8).

It's also interesting to note that the building block models have more interesting features, such as dips and distinct minima. This indicates the building blocks are impacting the bias of the functional linear model, in this case beneficially, and not simply adding additional freedom.

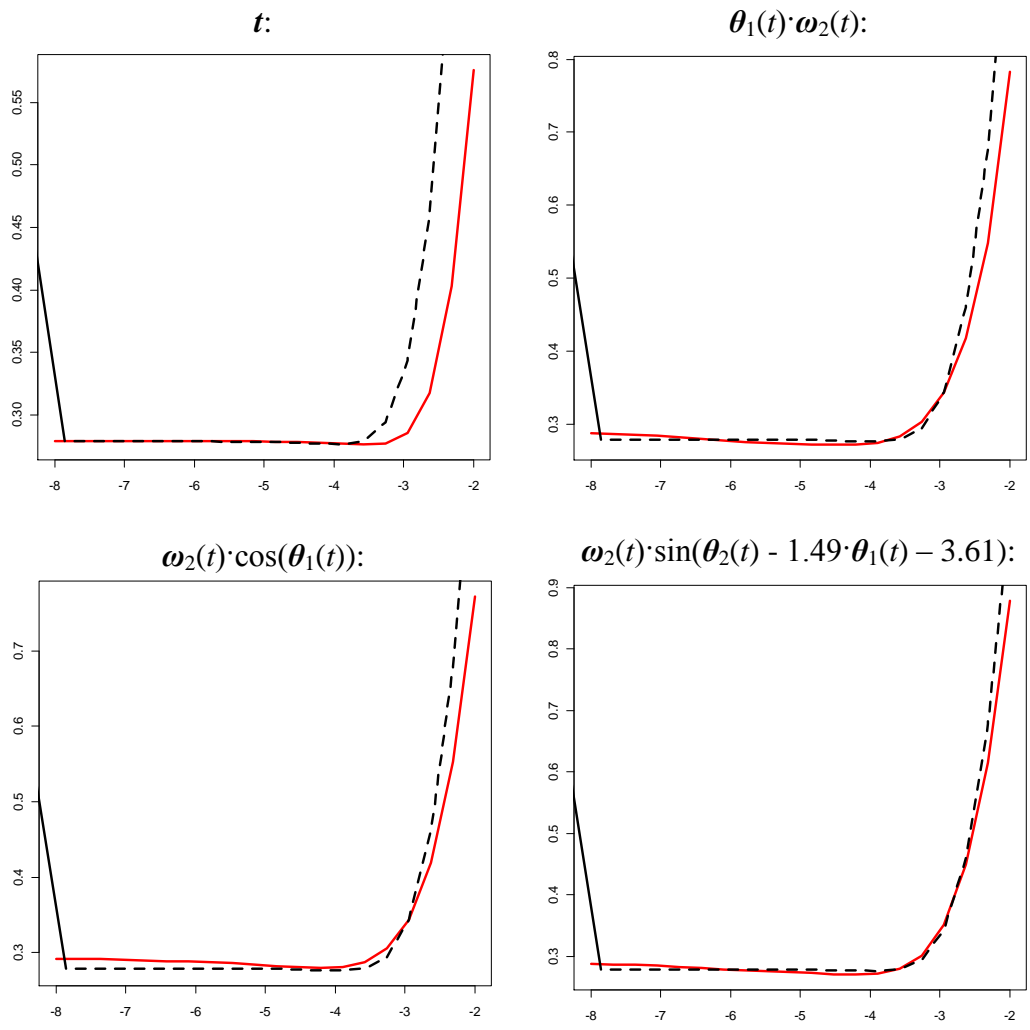
We can also examine the cross-validation curves for poor building blocks (Figure 18.9). These building blocks were also generated by the modeling algorithm as



**Figure 18.8.** The cross-validation error of the functional linear model using various building blocks for  $D^2\theta_2(t)$ , (red lines) shown in Fig. 8 and the null functional linear model (dashed black lines) versus the smoothing penalty  $\lambda$ . In each case the null model is dominated by the building block model for all coefficient smoothing penalties.

alternative models of different complexity. In these sweeps, the cross-validation fails to satisfy either reaching a lower minimum error or tolerating a higher smoothing penalty than the null model.

Additionally, the curves have very few or weak features, suggesting they are not introducing much bias to the model. In some cases, the performance is



**Figure 18.9. Cross-validation error of several poor building blocks (solid red) and the null model (dashed black). The poor building blocks fail to either achieve lower minimum error or tolerate higher smoothing penalties. The poor building blocks, other than  $t$ , were also building blocks generated by the modeling algorithm.**

indistinguishable from the null model. The method also rejects several other building blocks hypothesized by other generated models.

It's important to note that the smoothing on the building blocks needs to be as loose as possible. For example, when building a basis for  $\cos(\theta)$ , it is okay to heavily smooth  $\theta$ , but since the cosine makes the signal more complex, we do not want to require as

much smoothing on the building block expansion. Otherwise, the distinction between useful and poor building blocks diminishes. I believe this occurs because a heavily smoothed building block does not substantially change the bias of the linear model, and therefore simply increases the flexibility of the linear model. In this case the building block cross-validation curves appear as copies similar to the null model but shifted to the right at higher smoothing penalties.

Finally, there are some important limitations to this approach. First of all, it cannot distinguish perfectly the useful building blocks from all possible building blocks. This depends on the data available for making the distinction and also the fact that there exist many approximations to the building blocks of the system.

However, these results show that this procedure *can* pick out very useful building blocks, and reject very poor building blocks. Additionally, we may be able to improve the resolution with additional data.

## **Conclusions**

In this chapter, I explored detecting building blocks of nonlinear systems using FDA. Building blocks are small subcomponents of a full mathematical model such as nonlinear terms. The ability to detect a building block enables us to model and test individual components of a system, such as individual reactions occurring in a cell, without having to find or build an entire system model. Or alternatively, it could be used to build the complete model of a complex system incrementally, allowing the model to scale into high-dimensional domains.

The proposed method builds a functional linear model using a candidate building block expression of the system. The basis expansions of the linear model's coefficients mimic (or fill-in) the model components that surround the building block, so that we

can judge the merits of the building block without knowing the full model ahead of time. A useful building block is defined as having a building block functional linear model that both reaches a lower cross-validation error and tolerates higher smoothing penalties, spiking in error later in the sweep, than the null linear model. Poor building blocks fail to meet one or both of these requirements.

I examined this approach on a dataset of the dynamics of an insect wing during hovering flight. I used a custom registration procedure for slicing the time series into scaled periods of the wing's flapping. The adjusted scaling of the periods after registration showed a slow drift in the period length that appears to correlate with the lateral drift of the insect's position over time. Further analysis of the dataset showed small drifts in the wing angles over the consecutive periods and covariances between different peaks in the period. However, much of this variation was explained by a linear functional model of the period number. It is likely that these variations are a combination of noise and slight drift and are unimportant to understanding the dynamics of the wing itself.

I used building blocks from several models of this data generated from an equation search algorithm. The building block method using functional linear models identified four promising building blocks for the generated full models, while rejecting many other building blocks from alternative generated models. The successful building blocks dominated the cross-validation scores of the null model.

Based on this result, the building block method might be a useful method for validating a complete model as well. The building blocks of a full model that accurately represents the systems basic physical mechanisms should all individually be useful for explaining the data in a functional linear model.

### **Summary**

A major challenge in interactive evolution is extracting user preferences with minimal probing. We introduce an interactive multi-objective co-evolutionary algorithm that actively selects the most informative probes: We simultaneously co-evolve a population of candidate models that explain users' selection so far, and a population of candidate probes that cause the most divergence among model predictions, thereby elucidating model uncertainties (divergence). As progress is made, we begin selecting for probes with the highest expected outcome averaged among different models, thereby exploiting model certainties (consensus). In the evolution of pen stroke drawings, we find this technique to be highly effective at extracting preference models from very limited human interaction. Using only pair-wise preference questions, strategy and preference in pen stroke drawings are extracted in fewer than ten user probes. Our results show that the optimal questions to probe the user need not include drawings similar to the target drawing. Instead, the user models converge on trends in the user responses, thereby extrapolating strong preference for target drawings which the models are never actually trained to prefer.

### **Introduction**

Interactive evolution is a powerful explorative search technique that utilizes human input to make subjective decisions on potential problem solutions (Dawkins 1996; Takagi 2001). The fitness landscape in each domain is thereby determined explicitly by the human user. Reliance on human input however, induces two major challenges: First, the time cost to collect human input greatly prohibits the discovery of complex solutions. Second, the quality and accuracy of human input greatly degrades with repeated prompts for input.



In this chapter, we introduce a co-evolutionary algorithm to maximize the information obtained from the user and minimize the necessary interaction. We co-evolve a population of individual solutions with a population of models that predict the user's preference. Co-evolved solutions are used both to maximize user preference, and also to probe the user in order to refine uncertainty in the user models, two objectives that are not necessarily aligned.

Our primary hypothesis is that intelligently probing the user for input based on their co-evolutionary behavior can generate more accurate user models than conventional modeling from very limited user interaction. New user probes must challenge and refine uncertainty and ambiguity in the model population.. We claim that – like the game of *20 questions* – the co-evolved individual solutions provide invaluable information to select these new user probes and find optimal user questions base on answers to previous probes.

To further simplify the interaction with the user, all probes for input ask the user for a single preference decision between two individual solutions (e.g. drawings). Correspondingly, our user model encoding is a comparator which predicts preference between pairs of solutions. This is also a very natural design for a human's preference. Decisions about how preferable an individual is must be made in the context of another individual. Although this mechanism cannot assign fitness values, evolution can utilize it effectively in selection and ranking.

The key goal in this chapter is to extract accurate user models through minimal user interaction. Human users can only answer a small number of questions before becoming “numb” to prompts (Hollnagel 1993). Therefore, we perform experiments which evolve pen stroke drawings using ten or fewer user prompts. The accuracy is

evaluated based on the user's preferred target drawing.

We then compare our results with the interactive costs of a random search and a perfect local search algorithm. The random search comparison shows how effectively the exponential domain can be narrowed through co-evolution. The perfect local search comparison, where the user essentially draws their exact preference explicitly, shows how the interactive co-evolution of user models and probes algorithm can extract a specific preferred drawing from the user with fewer user probes and much simpler binary preference questions.

### **Related Work**

In this work we utilize genetic programming where a population of potential problem solutions is evolved in a Darwinian fashion (Koza 1992). We also utilize co-evolution, where two or more populations are evolved which directly or indirectly impact the evolution of each other in order to improve final solutions (Hillis 1991) (Zykov, Bongard et al. 2005). In traditional interactive evolution, a human user is presented with one or more candidate individuals being evolved for selection. The human user directly performs selection and favored individuals are selected for propagation of offspring into the next generation (Poli 1996; Takagi 2001).

A new area of research involves partial human interaction. In these algorithms, the user provides constraints on the problem to narrow traditional evolutionary search (Poli and Cagnoni 1997). A very promising area in interactive evolution research is agent based modeling. In this research, many user models are learned through interaction in order to study their collective behavior (Bonabeau 2002; Ihsan, Eric et al. 2005). In this chapter, we investigate how to discover the most optimal user models with the minimal amount of user interaction through co-evolution.

In user preference modeling research, a candidate critique agent (CCA) is trained to estimate the favor for single individual solutions. The CCA learns weights on individual parameters to give exact fitness values (Bishop 1996; Linden, Hanks et al. 1997). The weights introduce constraints on individuals and hence the CCA is very similar to partial interaction techniques.

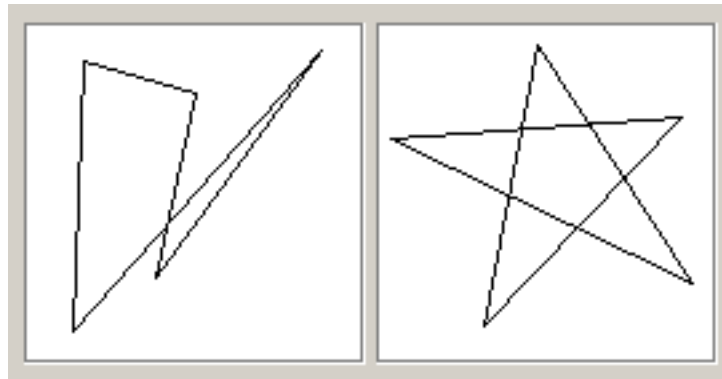
Our co-evolutionary design is based on an estimation-exploration algorithm (EEA) setup. Unlike classical evolution, an EEA consists of three components: A population of estimators, a population of exploratory solutions, and a target hidden system (Larrañaga and Lozano 2002; Zykov, Bongard et al. 2005). In this case the evolving problem solutions comprise the exploratory population, the ensemble of comparator models are the estimation population, and the abstract fitness landscape of the user's preference is the target hidden system.

## **Fitness of Comparisons**

### *Comparison vs. Fitness*

In this chapter, we utilize a pair-wise preference model as the basic element to define the complex fitness domains explored in interactive evolution domains. Subjective selection must be done in the context of one or more other individuals to have meaning (Figure 19.1). We claim that the decision between two individuals is a fundamentally easier and more natural decision than assigning precise preference values to single individuals.

*Which drawing is more interesting?*



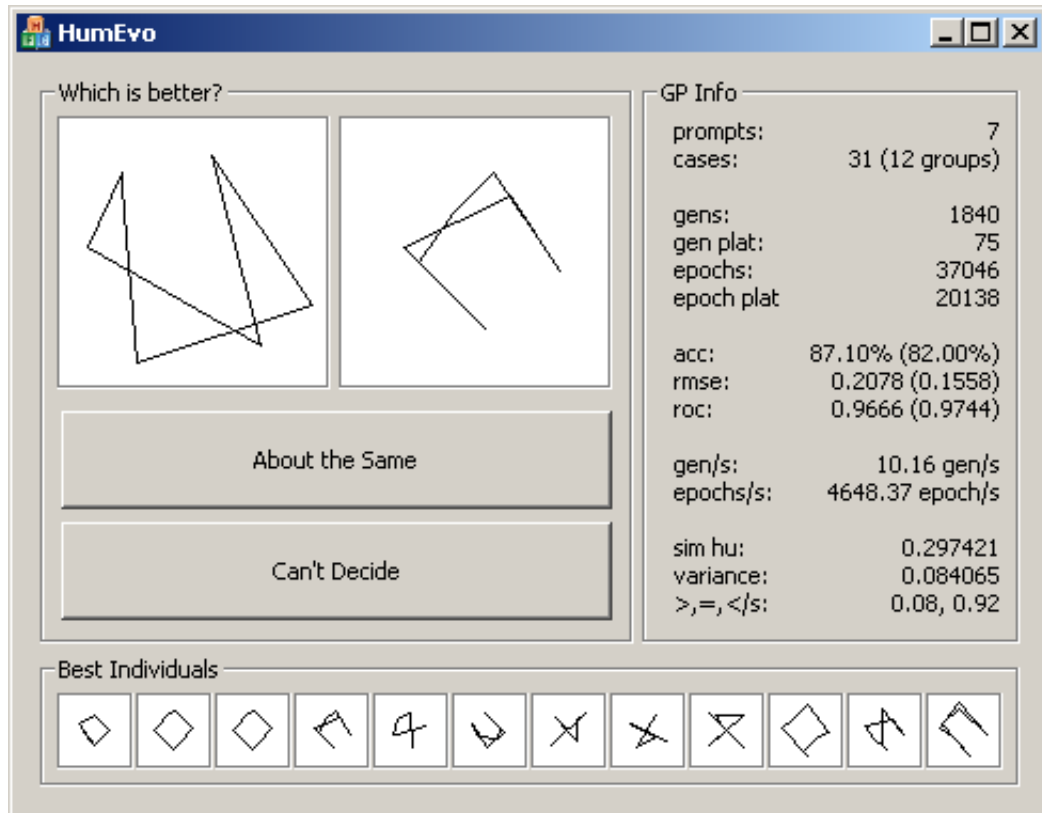
**Figure 19.1. Example comparison between two pen drawings.**

Prompting the user to make individual comparison has proven very effective in other interactive evolution techniques such as incorporating human comparison into tournament selection (Poli and Cagnoni 1997). In this chapter, we use the comparisons indirectly by co-evolving comparator models. The model population is used to evolve individuals and the user's input does not perform explicit selection.

### ***User Interaction***

The user interface presents two individuals to the user, and asks to click which one is more preferable, or optionally select them to be equally preferable, or for debugging purposes, request a new pair. An example of the screen shown to the user during runtime is shown in Figure 19.2.

The right-most panel displays performance and debugging information for expert users. The bottom-most panel shows the current best individuals in the population produced by the comparator model of the user.



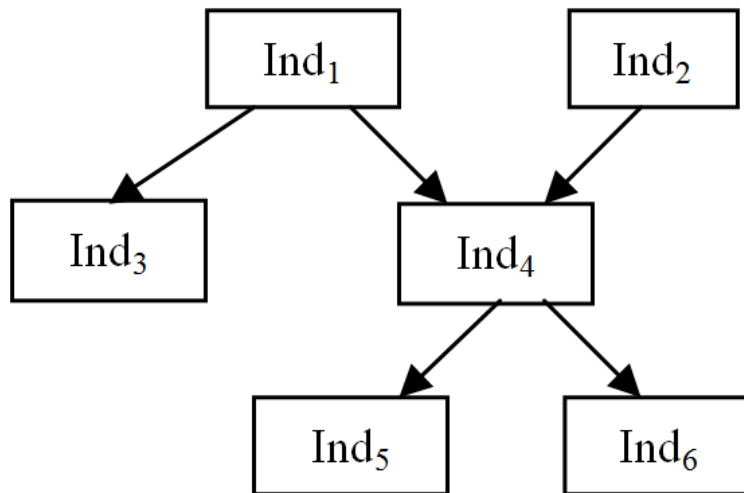
**Figure 19.2.** The user interface presented to the user.

### *Preference Relation Graphs*

#### *The Relation Graph*

Comparator models define a fitness landscape through cascading many comparisons. For example, if  $ind_A$  is better than  $ind_B$  and  $ind_C$  is better than  $ind_A$ , it follows logically that  $ind_C$  is also better than  $ind_B$ . Therefore, relations between each individual can be thought of as a directed graph.

It is important to note that comparison relations are prone to cyclical reasoning. For example,  $ind_A$  could be better than  $ind_B$ ,  $ind_B$  better than  $ind_C$ , and  $ind_C$  better than  $ind_A$ . We avoid this by storing all comparisons as an acyclic tree.



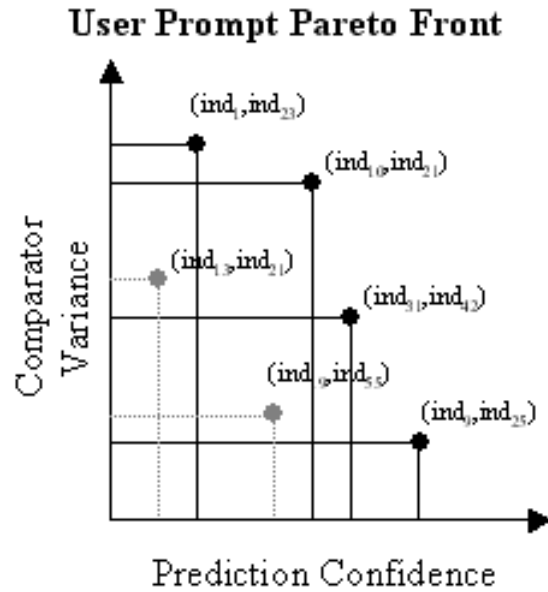
**Figure 19.3. Example comparison relational graph of six individuals.**

This tree in Figure 3 contains six individuals stored after five prompts to the user, where arrows indicate each better-than response. Note that there are nine relations that can be derived from this graph. The first five are the arrows shown. The next two are Ind<sub>1</sub> better than Ind<sub>5</sub> and Ind<sub>1</sub> better than Ind<sub>6</sub>. The last two are Ind<sub>2</sub> better than Ind<sub>5</sub> and Ind<sub>2</sub> better than Ind<sub>6</sub>.

To continue growing the relation tree, each prompt to the user contains one individual already in the tree, and one individual from the current population. The main advantage of this technique is it provides the potential for the number of known relations to grow at a binomial rate with user prompts in the ideal case, and a linear rate in the worse case.

#### *Minimizing User Prompts*

A large problem in interactive evolution we address in this chapter is the user becoming over worked. A user often becomes “numb” to new prompts after a while, meaning they begin to put less thought into their selections and produce noisy data (Hollnagel 1993). To reduce this effect, we take large strides to minimize the



**Figure 19.4. Example Pareto front plot of eight potential comparison pairs.**

necessary interaction with the user. We apply active learning to maximize the potential information gained in each new prompt to the user. The goal when presenting a new prompt is for the response to refine the current comparator model and refine uncertainties and generality.

In the co-evolutionary setup, we train an ensemble of user models. The average of the ensemble then performs all selection in the individual population evolution (Kohavi 1995). We examine their separate predictions to measure their uncertainty and ambiguity. For example, different models in the ensemble may have strongly disagreeing predictions for different pairs. One may strongly predict a greater-than, while others may weakly predict a less-than. This is a case where feedback on a high variance relation will greatly improve the generality of the ensemble.

To select a new comparison prompt to the user, we consider two factors: the variance of the pair in the ensemble, and the strength of their predictions. In other words, we



**Figure 19.5. The basic structure of an individual comparator user model.**

are interested pairs that have highly different predictions that are very strong and perhaps overfit in the model. These two parameters form a Pareto front for prompt selection (Ficici and Pollack 2001).

The prompt selection is done by generating all pairs of individuals with one from the current relation tree and one from the current population. These pairs are then considered on a bivariate graph defined by their variances and prediction strengths.

The Pareto front consists of points that are non-dominated. This Pareto front favors pairs that are both high variance and strong predictions. In other words, comparators in the ensemble have strong differing opinions on the predicted outcome. Points on the front with low confidence and high variance correspond to ambiguous pairs that are unexplored areas of the prediction domain. The high confidence and low variance predictions correspond to pairs that are well defined cases which can be refined to higher detail.

## **Predicting Comparisons**

### ***Basic Comparator***

A simple comparator takes two individuals as input, and outputs three cases: better-than, equal, and worse-than. When comparing individuals however, we are only interested in their better-than or worse-than outcome. Therefore, the interface of the



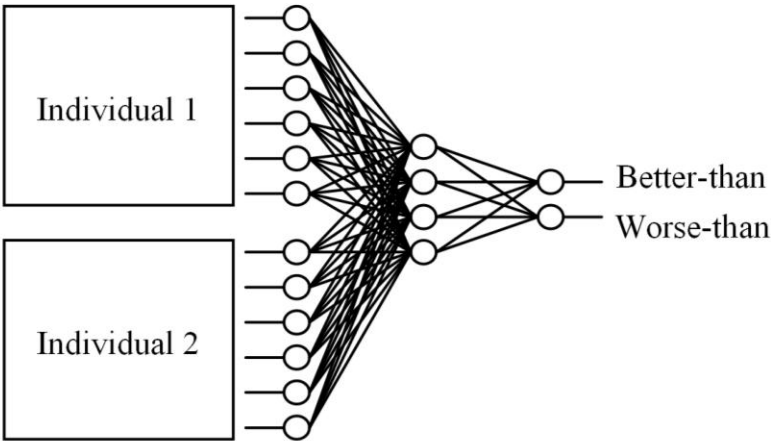
comparison predictor takes two individuals, and outputs two confidence values for better-than and worse than. This structure is shown below in

The two outputs provide confidence values for each case. Their difference yields the final outcome in selection, and strength in user prompt calculation.

*Neural Network Comparator*

Neural nets are a natural fit for most comparison user modeling. They have robust regression power with excellent interpolation and extrapolation characteristics (Cybenko 1992). Their classification output also corresponds to their statistical confidence in their prediction. In other words, noisy samples or conflicting samples reduce prediction confidence but in general maintain prediction accuracy (Hermann 1995). The basic structure of a comparator neural net is shown in Figure 19.6.

We use a network with a single hidden layer. We choose a number of hidden units sufficiently large enough for the domain. We then utilize early stopping on RMS with a eight-fold validation set to avoid over-fitting (Lawrence and Giles 2000).



**Figure 19.6. The structure of a neural network user model.**

## **Algorithm**

The interactive co-evolution of user models and probes algorithm presented in this chapter maintains three essential components: the individual population, the relations graph, and the comparator model ensemble. The algorithm operations on these components in five stages: calculating the best comparison pair, requesting the user's input, generating new relations based on the feedback received, training the comparator model ensemble, and evolving the individual population using the comparator model.

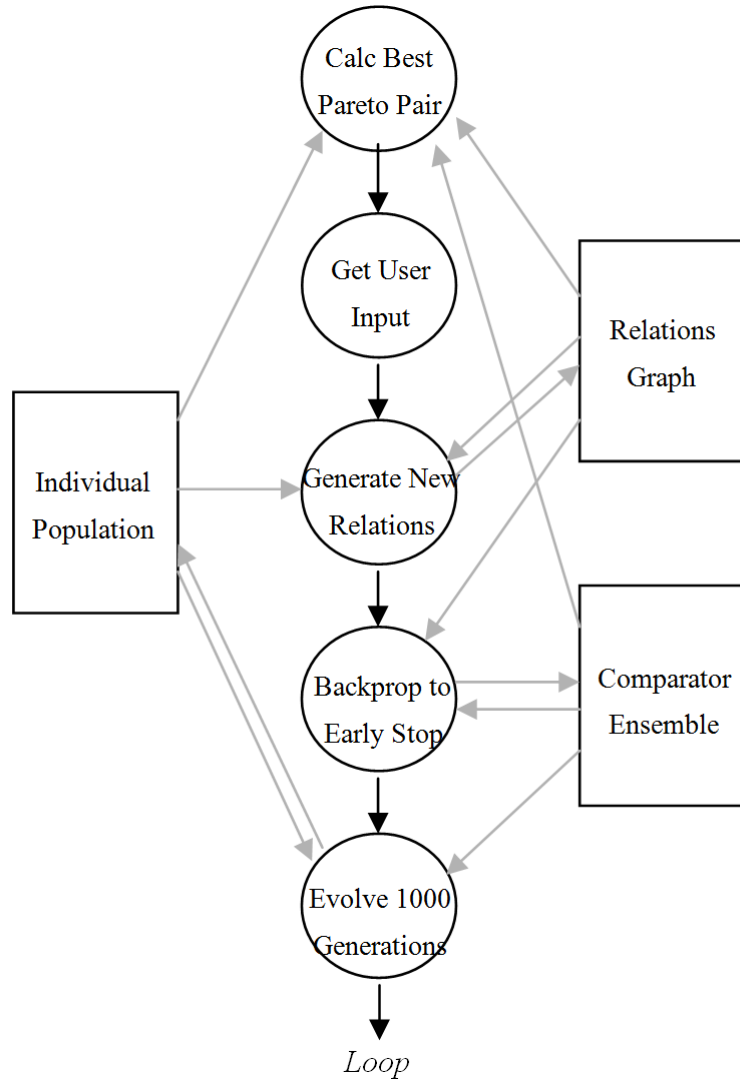
The algorithm consists of five stages that operate on the individual population, the relations graph, and the comparator model ensemble. The first stage chooses the best Pareto pair of individuals to present to the user as above. The algorithm then pauses for the user to respond. Next, all possible comparisons derived from the response when added to the relations graph are calculated. Then the comparator models are randomized and re-trained to their early stopping point. Finally, the individual population is evolved for one thousand generations before returning to stage 1.

The algorithm loops until the user is satisfied with the top ranked individual co-evolved by the comparator ensemble. As long as the user has a consistent preference, further iterations will stabilize and simply fine tune the preferred result.

## **Experiments Evolving Drawings**

### ***Drawing Encoding***

In this experiment we evolve drawings produced by a series of closed pen strokes. Each individual encodes each coordinate drawn to on a 32 by 32 pixel image. In our experiments we predefine the number of strokes for each drawing, but this could easily be evolved as well in future work.



**Figure 19.7. The comparator model based interactive evolution algorithm basic outline.**

The search space for these types of drawings increases exponentially with the number of pen strokes. The number of possible individuals for  $N$  pen strokes is calculated below.

$$\# \text{ of pixels} = 32 \cdot 32 = 1024$$

$$\# \text{ of possible individuals} = 1024^N$$

The large search space makes the discovery of preferable individuals an excellent

Table 19.1. Neural Network Training Parameters.

Parameter	Value
Hidden Layers	1
Hidden Units	32
Learning Rate	0.001
Momentum	0.5
Cross Validation Folds	8

application for an evolutionary algorithm.

### *User Model Encoding*

The comparison user model is encoded as a neural network as described in above. We choose to compare images based upon their image moments. Specifically, the zero and first order moments plus the first three Hu Invariant Moments, yielding a total of twelve inputs to each neural net.

The parameters for training the neural nets are summarized in **Table 19.1**. These parameters are chosen empirically and they perform well for this experiment. Although they can influence comparison performance in extreme cases, they generally only impact the training time required.

### *Evolutionary Settings*

We use standard genetic programming to evolve individuals in this experiment with the exception that all selection and fitness comparisons are performed using the

**Table 19.2. Evolution Parameters.**

<b>Parameter</b>	<b>Value</b>
Population Size	64
Selection	Deterministic Crowding using the comparator ensemble
Mutation Probability	0.05
Crossover Probability	0.75

trained comparator model ensemble. Individuals are not given explicit fitness values but are instead ranked using the averaged comparator ensemble. A summary of the evolution parameters is shown in Table 19.2.

We use Deterministic Crowding for selection because it is a natural fit for a pair-wise comparator. The Deterministic Crowding method maintains population diversity through child-parent elitism and tends to follow multiple divergent pathways to the final solution (Mahfoud 1995). This results in more variety in user prompt selection and better generalization of the comparator fitness landscape.

## **Shape Preference Results**

### ***Square Drawings***

The first experiment conducted tests the ability of algorithm to identify and infer a user’s preference for “square-like” drawings from initially random pen drawings. Each individual is encoded as a series of four pen strokes. In this experiment we compare

with random search and local search techniques. These comparisons gauge how effectively our algorithm reduces the interactive cost with the user to discover the target drawing.

To quantify the difficulty of this problem we calculate the probability of finding a “square-like” drawing through random search. Note that the square is uniquely determined by two of its vertices. Given that a “square-like” drawing can have any orientation or size, and we allow the 3<sup>rd</sup> and 4<sup>th</sup> vertices to have some noise of four pixels, the probability and expected random individuals observed,  $T$ , to find such a drawing is calculated below.

$$P_{square} = \frac{(\#loc_1) \cdot (\#loc_2) \cdot (\#loc_3) \cdot (\#loc_4)}{(\#pixels) \cdot (\#pixels) \cdot (\#pixels) \cdot (\#pixels)}$$

$$P_{square} \approx \frac{(32^2) \cdot (32^2) \cdot (4^2) \cdot (4^2)}{(32^2) \cdot (32^2) \cdot (32^2) \cdot (32^2)}$$

$$P_{square} \approx \frac{4^4}{32^4} = \frac{1}{4096} \approx 0.024\%$$

$$\therefore E(T_{square}) \approx 4096$$

Therefore, the probability randomly generate a square is 0.39% and the expected iterations to encounter a square in a random search is 256.

An alternative to interactive evolution is an interactive local search algorithm. In this technique, the user is given a random individual and asked to fine tune parameters individually. In pen stroke drawing this corresponds to adjusting the x and y coordinates of each pen stroke vertex to desired locations. This technique can be viewed as very simplistic partial interactive evolution where the user effectively constrains each parameter individually.

The local search algorithm we compare with can be thought of as a perfect algorithm

for transforming a random drawing into the target drawing. The user is assumed to be an oracle that makes perfect choices to optimally tweak parameters with the minimum number of prompts.

Here we calculate a lower bound on the expected number of local adjustment steps necessary to shape a random pen stroke drawing to a square shape. Recall that a square is determined by two vertices. So we begin by calculating the expected diagonal of the square from two random points on the 32 by 32 pixel grid.

$$\begin{aligned} \langle \Delta x \rangle \langle \Delta y \rangle &= \frac{\sum_{x_1=0}^{31} \sum_{x_2=0}^{31} |x_1 - x_2|}{32^2} = \frac{10912}{32^2} \approx 10 \\ \langle d \rangle &= \sqrt{\langle \Delta x \rangle^2 + \langle \Delta y \rangle^2} \approx 15 \end{aligned}$$

Next, the expected mean of these two points, and all random vertices is the center point of the grid ( $x=15, y=15$ ). This means that the two remaining points are expected to be at the center of the final square. Hence, they must each move half a diagonal,  $\langle d \rangle / 2$ .

In the easiest case, these points move only along an individual axis. In the worst case, they move diagonally, stepwise on the grid. The expected number of steps per vertex is calculated below.

$$\langle steps \rangle = \frac{1}{16} \cdot \sum_{x=0}^{15} x + \left\lceil \sqrt{15^2 - x^2} \right\rceil \approx 18$$

Therefore, the lower bound on the total number of expected local updates to the x and y coordinates of the two remaining vertices is approximately 36.

Note that this is a lower bound approximation on the expected minimum number of

Prompt	Comparison		Resulting Top Three Guesses		
1					
2					
3					
4					
5					
6					

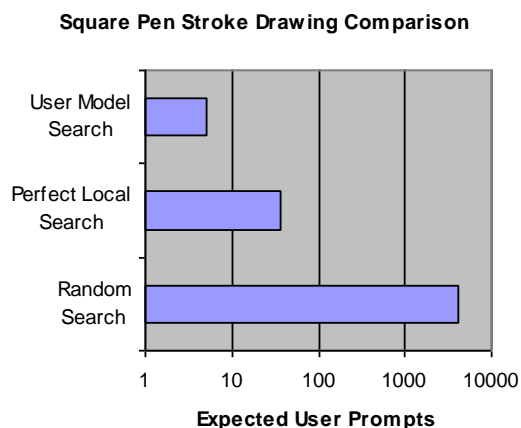
**Figure 19.8. The prompts given to the user and the resulting top three guesses over six iterations.**

user prompts required for a perfect local search algorithm to form a pen stroke drawing of a square at any orientation. An expert user is required to identify a desired diagonal and then isolate the remaining vertices accordingly.

Figure 19.8 shows a standard run for a square drawing using the comparator user model interactive evolution algorithm. The user has a specific strategy to prefer shapes with parallel sides and right angles consistent with a square. The user's preferred drawing for each comparison is shown in green, non-preferred drawings are shown in red, and drawing deemed to be equivalent are shown in dark yellow. This is a very representative run since the initial random prompts do not include any box-like drawings. If by chance they do, the runs tend to converge on a box shape immediately.

First, notice that none of the comparisons shown to the user involve a square. Based





**Figure 19.9.** The number of user prompts expected between the compared algorithms to find a square shape.

upon the several non-square comparisons, the comparator model is able to infer that the user is likely to prefer a square shape. If in fact the user was not knowingly seeking squares but giving feedback on some unknown preference, the algorithm would predict and identify box-like solutions they are likely to favor.

The algorithm successfully found a square in its top three guesses after four user prompts, and ranked a square as its top guess after six prompts. This is a vast improvement over the random search which is expected to require 4096 user interactions before finding an approximate square.

Figure 19.9 compares the user comparator model algorithm with the perfect local search and random search algorithms. The logarithmic scale shows that the user comparator model makes significant improvement over the perfectly performing local search and vastly reduces the search cost over random search.

### ***Star Drawings***

In this experiment, the algorithm evolves drawings with six pen strokes. Here we evaluate the algorithm ability to identify a preference for star-shaped drawings.

We quantify the difficulty of this problem by calculating the probability of finding an approximate star shape randomly. To approximate the number of star shapes possible, we split the drawing space into six regions: a center where no vertices can be, and five surrounding regions for each vertex, all of equal area. Note that a five point star is uniquely determined, by region in this case, but three of its vertices. The probability and expected random individuals observed,  $T$ , to find an approximate star shaped drawing is calculated below.

$$P_{star} = \frac{(\#loc_1) \cdot (\#loc_2) \cdot (\#loc_3) \cdot (\#loc_4) \cdot (\#loc_5)}{(\#pixels) \cdot (\#pixels) \cdot (\#pixels) \cdot (\#pixels)}$$

$$P_{star} \approx \frac{\left(\frac{5}{6} \cdot 32^2\right) \cdot \left(\frac{2}{6} \cdot 32^2\right) \cdot \left(\frac{1}{6} \cdot 32^2\right)^3}{(32^2) \cdot (32^2) \cdot (32^2) \cdot (32^2) \cdot (32^2)}$$

$$P_{star} \approx \frac{5}{3888} \approx 0.128\%$$

$$\therefore E(T_{star}) \approx 777.6$$

Therefore, the probability to randomly generate a star drawing individual is 0.128%, and the expected number of random generations to encounter one is 777.6. Note that this is an easier search than finding a square since we are not requiring specific angles, so the acceptable star shapes could be quite deformed.

Following the same logic as above, we now approximate a lower bound on the expected user necessary to form a star pen stroke drawing.

As found earlier, the expected distance between two random vertices on the 32 by 32 pixel grid is approximately 15. To simplify this calculation we make an approximation that the expected locations of the other three vertices lay at the center of the star. The radius of this star with line length 15 is easily calculated to be approximately 8. The expected number of steps, moving both in x and y is calculated below.

Prompt	Comparison		Resulting Top Three Guesses		
1					
2					
3					
4					
5					
6					
7					

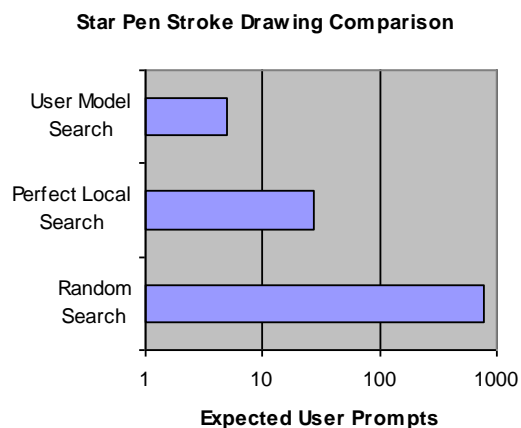
**Figure 19.10.** The prompts given to the user and the resulting top three guesses over seven iterations.

$$\langle steps \rangle = \frac{1}{9} \cdot \sum_{x=0}^8 x + \left[ \sqrt{8^2 - x^2} \right] \approx 9$$

Therefore, the lower bound on expected local updates of the three remaining vertices is approximately 27.

Again, that this is a lower bound for a perfect local search algorithm to form a pen stroke drawing at any orientation. A user would need to identify a starting edge and then tune remaining vertices accordingly.

Figure 10 shows a standard run to evolve an approximate star shaped drawing. The user has a general strategy when answering comparison prompts to prefer shapes with



**Figure 19.11. The number of user prompts expected between the compared algorithms to find the target star shape.**

multiple sharp pointed corners such as a star shape has. Preferred draws are shown in green, non-preferred in red, and equivalent drawings in dark yellow. This is a representative run since the initial random prompts are very dissimilar to the target star shape. In the unusual case, early prompt may resemble the target shape by random chance, resulting in nearly immediate convergence to the star shape.

Notice in Figure 10 that a very well formed star is derived as the top predicted shape after five user prompts. No prior prompts required a star shape. Instead the comparator model inferred the star shape as an optimum solution given the user responses favoring shapes with multiple sharp edges.

The next three prompts answered by the user are shown to demonstrate the comparator model is stable and continues to favor the consistent star-like shapes with further input. Therefore, the algorithm has likely converged on the star shape preference.

Figure 19.11 shows the comparison of the three algorithms. Again, the user comparator model makes substantial improvement over the perfectly performing local search and random search.

## Inferring the User's Fitness Landscape

### *Discovering a Clock Drawing Preference*

In this experiment we want to visualize the fitness landscape being learned by the comparator model. To do this, we modify our pen stroke drawing individual to have only a single pen stroke originating from the center of the drawing area. The search space of the individuals is now only a single coordinate,  $x$  and  $y$ . We can then display relative fitness values for all possible individuals in a 3D surface.

For this experiment, the single pen stroke encoding is considered to be a clock hour hand. The user is then asked to prefer clocks drawn where the time is closer to some time of day that they prefer. The user for the experiment chooses to prefer clock drawings where the hand points to either 3:00 or 7:00. Therefore, this experiment has two equally favour global maxima solutions.

### *Fitness Landscape of a Comparator*

To determine if the comparator accuracy learns the dual solutions in this experiment we need some way to calculate a fitness landscape from a comparator model. Therefore, we need to define a fitness calculation for a single individual using the binary comparator.

For this experiment, the search space for all clock drawings is the total number of pixels,  $32^2$ . For this small search space, it is feasible to list all possible drawings and compare them with a particular drawing. We calculate an effective fitness by taking the average confidence of better-than and worse-than outcomes when compared with all possible  $32^2$  clock drawings.

$$Fitness = \frac{1}{32^2} \sum_x \sum_y Better(Ind(x, y)) - Worse(Ind(x, y))$$

In this expression, Better() and Worse() are the two confidence outputs of the basic comparator model described above.

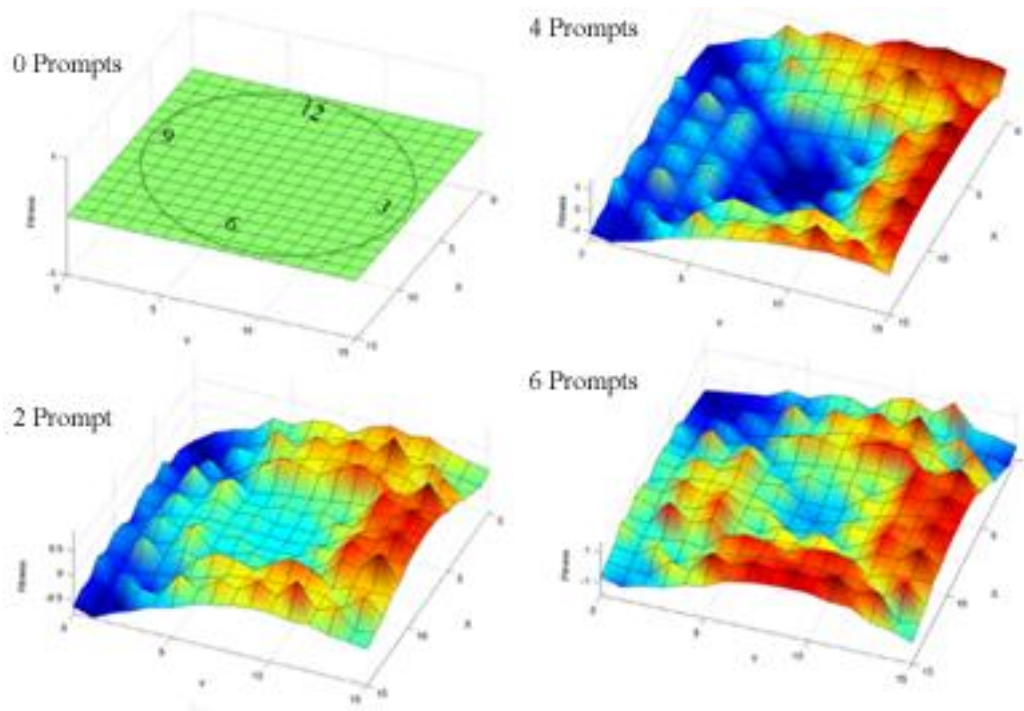
### ***Landscape Results***

Figure 19.12 shows four fitness landscapes of the comparator model over six user prompts. At zero prompts, the fitness landscape is entirely flat.

After two prompts, the comparator model has identified a single preferred clock time region, 3:00. It also strongly disfavors clock times between 7:00 and 11:00. After two further prompts, the comparator now favors times between 1:00 and 5:00. This appears to be an intermediary stage where the comparator has learned to favor clock hands of a minimum length. This is clear by noticing the fitness landscape shows very low fitness for short clock hands near the center, and high fitness for hands at the extremities between 1:00 and 5:00. Finally, after six prompts the comparator has successfully identified the two preferred regions near 3:00 and 7:00.

Notice that the fitness values on the xz-plan resemble an arch which peaks at approximately the clock hand 3:00 position. Correspondingly, the yz-plane fitness values exhibit the same phenomenon although slightly less accurately near the 7:00 clock hand position. This final resulting landscape shows the comparator has inferred a very friendly fitness landscape for the evolutionary search. Very gradual gradients exist near the target clock hand locations that should be easily descended.

It's interesting to note that the comparator landscape also exhibits a few other medium fitness clock hand areas such as near the 9:00 position. This is a weakly favored region that the algorithm shows some probability for preference in. Since these are local maxima, evolution is likely to focus on these areas. The Pareto criteria hence is likely to prompt the user to refine these regions in additional iterations.



**Figure 19.12. The clock time fitness landscapes calculated over six user prompts.**

## Conclusions

Experiments in this chapter show the interactive co-evolution of user models and probes algorithm to be effective at extracting preference models and resulting solutions using only limited human interaction. The approach used two-image preference questions to extract strategy and preference in pen stroke drawings in fewer than ten questions.

In comparison to the perfect local search algorithm, where the user essentially draws their exact preference explicitly, the interactive co-evolution of user models and probes algorithm requires roughly ten times fewer total user input. Furthermore, the prompts to the user are presented in much simpler binary preference questions.

Finally, our results showed that the optimal questions to the user need not include drawings similar to the target. Instead, the user models extrapolated preference for target drawings which the models are never explicitly trained to prefer.

## CHAPTER 20. PUBLIC GOODS GAMES

### **Summary**

This chapter proposes empirical models of an individual's behavior in the Public Goods game and analyzes their predictions. These models were extracted directly from experimental data using an automated algorithm – the main topic of my thesis research – that searches for the simplest empirical formula that model key dynamics in the dataset. An interesting feature that arose repeatedly across several models was the individual's cumulative earnings when compared relative to the group, suggesting that players behave differently depending on their overall success. The inferred models also suggest that individuals adapt their behavior based on the mean behavior of the group which tends to improve their cumulative earnings.

### **Introduction**

In this chapter, I apply symbolic regression to model individual behavior in the Public Goods Game (Hardin 1968). Public Goods Games are of particular interest in studying behavior or economics, as they test the choices people make to cooperate for greater benefit. I analyzed experimental data from the research of Jessie Barker, Pat Barclay, and Professor Kern Reeve that is yet to be published (only the results of my analysis are discussed here). This research considered two variations of the game: the standard game where players decide whether or not to cooperate and contribute to the public good, and the game with tug-of-war competition where players must also choose how much to spend to increase their share of the public good.

In the remaining sections, I briefly introduce background on the automated modeling approach, the methods and variables considered in the models, the modeling results, and finally discussion and conclusions.



**Table 20.1. Model symbol definitions.**

<b>Symbol</b>	<b>Meaning</b>
$n$	Round number
$x_n$	Fraction invested in the <b>public good</b> by an individual in round n
$y_n$	Fraction <b>kept</b> by an individual in round n
$z_n$	Fraction invested in <b>competition</b> by an individual in round n
$q_n$	Cumulative <b>earnings</b> of an individual in round n

### **Background**

The following experiments apply symbolic regression to the Public Goods Games data. See the description in the section “Symbolic Regression” on page 4 for more details on this technique.

### **Methods**

The experimental data contains five key variables of the behavior of the players: the round number, the contribution to the public good per round, the amount kept per round, the amount invested in competition (for tug-of-war) per round, and the total cumulative earnings.

In analyzing this data, I normalized all money amounts to fractions of the player’s total money allotted per round. The symbols for these variables and their meanings are shown in **Table 20.1**. Secondly, I calculated the mean fractions across the groups of

**Table 20.2. Model symbols for group averages.**

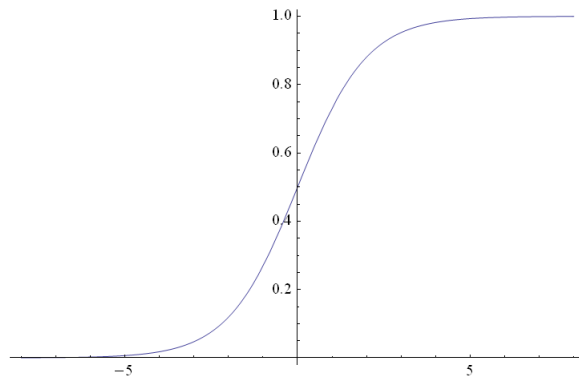
<b>Symbol</b>	<b>Meaning</b>
$\text{group}(x_n)$	Mean fraction invested in the <b>public good</b> by the group in round n
$\text{group}(y_n)$	Mean fraction <b>kept</b> by the group in round n
$\text{group}(z_n)$	Mean fraction invested in <b>competition</b> by the group in round n
$\text{group}(q_n)$	Mean cumulative <b>earnings</b> of the group in round n

players per round. These variable symbols are listed in Table 20.2.

In this chapter, I am interested in modeling how players choose to invest and allocate their allotted money per round. We can construct several different types of models using these variables. For example, perhaps strategies just drift over time, or perhaps they depend on outcomes of the previous round.

I attempted searching for models using several different sets of variables and different types of outputs. However, I found the most parsimonious and interpretable models when modeling the fractions each player will invest in the next round, as a function of how they and the group played in the previous round.

Because we are modeling a fraction value, I imposed a special constraint on the model using a sigmoid function. I forced all models to be inside of a logistic equation. The logistic equation (Figure 20.1), squashes values between zero and one, which producing effective fraction values.



$$\text{logistic}(x) = \frac{1}{1 + e^{-x}}$$

**Figure 20.1. The Logistic Function squashes all inputs to the range from zero to one. This function is used in the model structures because we are modeling a fraction value (e.g. the fraction of money invested in the public good). The input to the logistic function then has an interpretation of a strength toward zero (large negative values) or one (large positive values).**

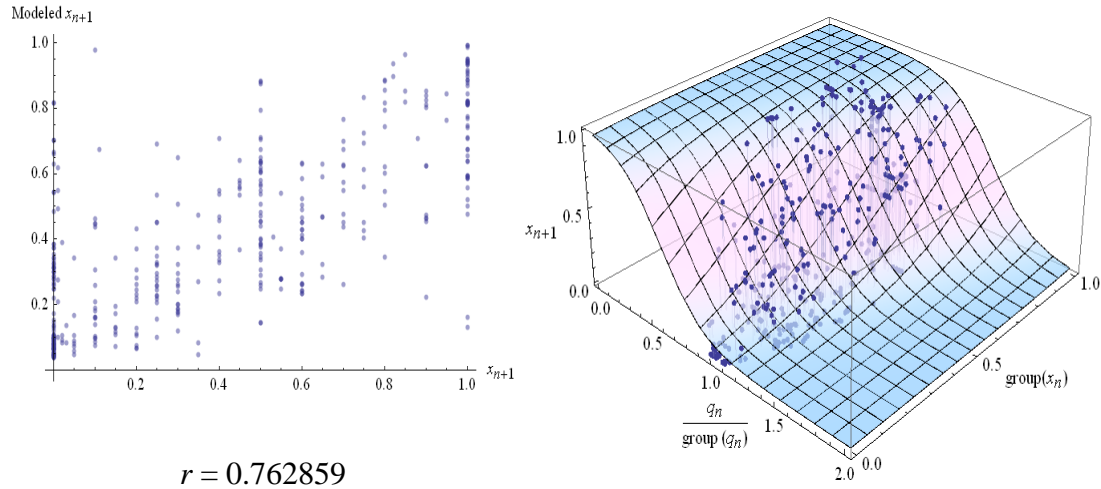
Using the logistic function constraints, we are now searching for equations that go into the squashing function. This changes our interpretation of the resulting formula. Instead of modeling the signal directly, they model a strength or tendency for the fraction to tend toward zero or one – for example, a desire to contribute to the public good, or a desire to defect.

## Results

Results are split into two sections: the model of the normal Public Goods Game, and the tug-of-war version with competition.

The modeling algorithm actually produces a list of potential models for any given dataset that span a range of model complexities. In this section, I have manually picked out the most parsimonious model in this list that picks up the most of the variance in the data.

**Model:**  $x_{n+1} = \text{logistic} \left( \alpha - \beta \frac{q_n}{\text{group}(q_n)} + \gamma \text{group}(x_n) \right)$



**Figure 20.2.** The model obtained for an individual’s contribution to the public good ( $x$ ) in the normal Public Goods Game. The left pane shows the correlation of the model predictions with the data. The right pane shows the predictions of the model (the 3D surface) next to the experimental data (the blue dots). The model suggests that a player is less likely to contribute if they have high cumulative earnings relative to the group, however they are more likely to contribute if others in the group contribute. The fitted parameters are  $\alpha = 3.51853$ ,  $\beta = 6.21075$ ,  $\gamma = 5.08922$ .

**Normal Public Goods Game**

In this standard version of the game, there is only one effective decision variable: how much to invest in the public good. The fitness change for a player in a single round of the game is given by:

$$w_n = 1 + 2 \text{group}(x_n) - x_n$$

The best model obtained is shown in Figure 20.2.

This model has two terms inside of the sigmoid. The first is a ratio of the cumulative earnings of the player to the earnings of the group. Interestingly, the player is less likely to contribute to the public good if they are earning more than average. The second term is the group’s mean contribution, which shows the player is more likely to

contribute if the group is contributing. The balance of these two factors appears to determine whether the player will contribute or not.

We can write a general condition for players to contribute to the public good ( $x$ ) for the normal Public Goods Game by solving for the threshold value analytically:

**Likely to contribute to public good if:**

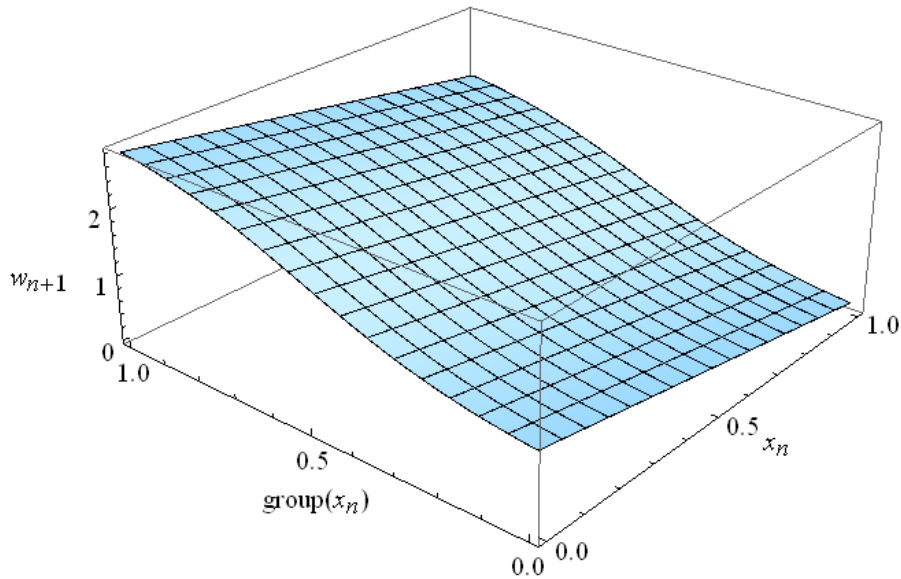
$$group(x) - 1.22 \left( \frac{q}{group(q)} \right) > -0.69$$

This predicts that if the group's mean contribution is sufficiently high, the player is more likely to contribute to the public good. Similar modeling of the  $y_{n+1}$  signal produces a nearly identical model, but with the sign of the coefficients reversed and with slightly different magnitudes.

Next, let's assume that this model captures the mean behavior of the group, and examine the fitness landscape for an individual player against the field. To do this, we can substitute the modeled  $x_{n+1}$  in for the mean  $group(x)$  behavior in the fitness function. Additionally, let's assume the player's cumulative earnings are the same as the group ( $q \approx group(q)$ ). The fitness function is then:

$$w = 1 + 2 \text{logistic}(-2.69 + 5.09 \text{group}(x)) - x$$

Solving the derivative with respect to  $x$  equal to zero, and the substituting  $x$  for  $group(x)$  indicates that the fitness has an analytical fixed point at  $x^* = 0$ . The Eshel's test (double partial derivative) yields zero, but we can also visualize this graphically:



**Figure 20.3. The fitness landscape for an individual against the field assuming that the group behaves according to the model. The individual’s fitness improves for increasing group contribution ( $group(x)$ ) and decreasing individual contribution ( $x$ ). Thus, the model predicts that players will contribute less and less, tending toward zero contribution.**

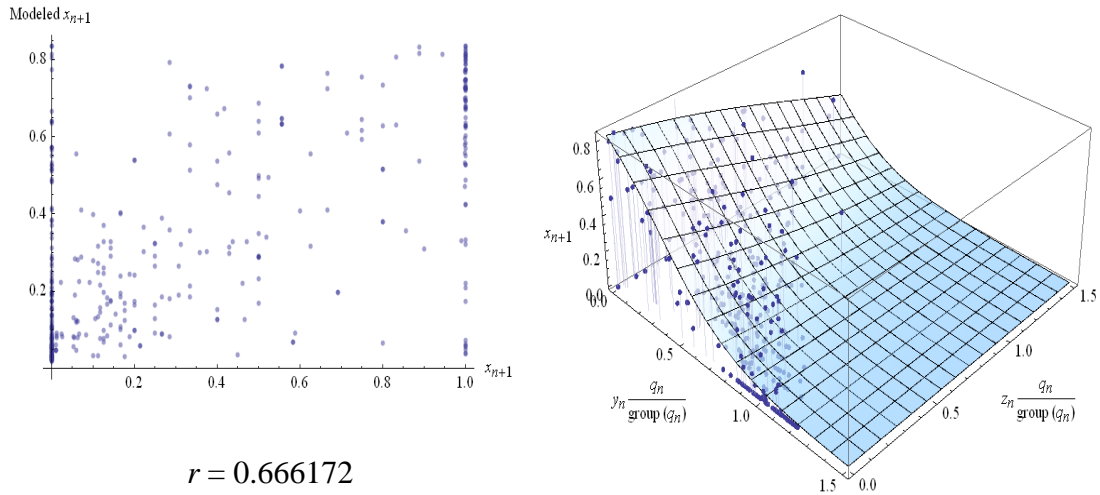
The effective fitness landscape for an individual, assuming the inferred model is shown in Figure 20.3. The individual can only move along  $x_n$ -axis, and can improve fitness by contributing less. However, contributing less also decreases the mean group contribution. Therefore, the model predicts that all trajectories of successive rounds played on this surface tend toward zero contribution by all players, assuming that the players play rationally. Therefore, the fixed point is stable.

### ***Tug-of-War Public Goods Game***

In this version of the Public Goods Game, there are now three options: contribute to the public good, invest in the tug-of-war, or keep. Here, I looked for a model of each option. The fitness change for a player in a single round is given by:

$$w_n = 1 + 2 \text{group}(x) \left( \frac{z_n}{\text{group}(z_n)} \right) - z_n - x_n$$

**Model:**  $x_{n+1} = \text{logistic} \left( \alpha - \beta y_n \left( \frac{q_n}{\text{group}(q_n)} \right) - \gamma z_n \left( \frac{q_n}{\text{group}(q_n)} \right) \right)$

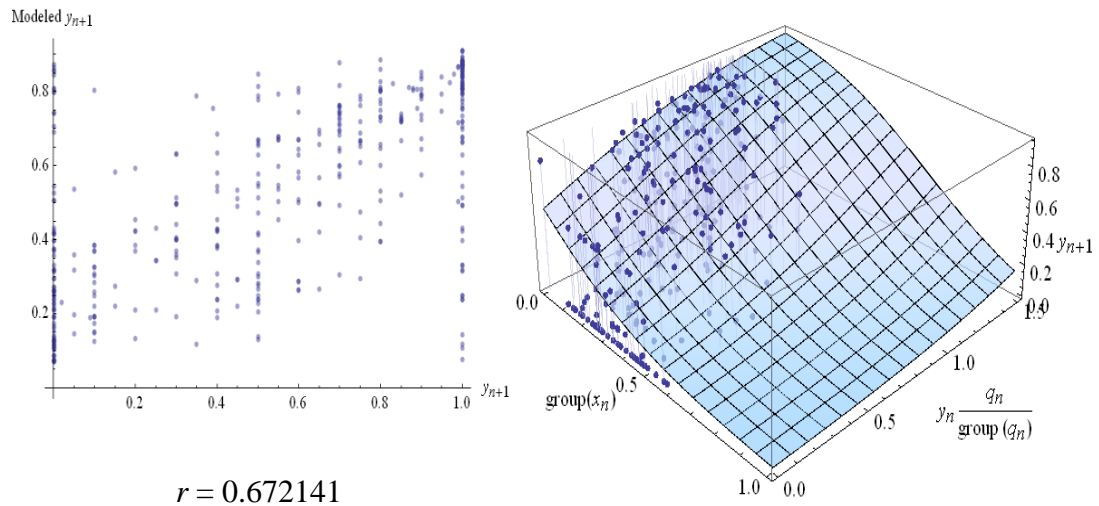


**Figure 20.4.** The model obtained for an individual’s contribution to the public good ( $x$ ) in the tug-of-war version of the game. The left pane shows the correlation of the model predictions with the data. The right pane shows the predictions of the model (the 3D surface), next to the experimental data (the blue dots). This model suggests if the player is doing well in cumulative earnings and kept and/or competed an amount previously, she/he is less likely to contribute. The fitted parameters shown are  $\alpha = 1.62552$ ,  $\beta = 4.61882$ ,  $\gamma = 1.37634$ .

The best model obtained for the contribution is shown in Figure 20.4.

Again, we see the ratio of the cumulative earnings to the mean of the group. We can interpret this model as if the player has been successful and kept ( $y$ ) or competed ( $z$ ) in the previous round, to continue not contributing to the public good. Additionally, the model indicates that the tendency to contribute ( $x$ ) arises from an ambient alpha term, suggesting that players may want to cooperate innately.

$$\text{Model: } y_{n+1} = \text{logistic} \left( \alpha - \beta \text{group}(x_n) + \gamma y_n \left( \frac{q_n}{\text{group}(q_n)} \right) \right)$$



**Figure 20.5.** The model obtained for an individual’s fraction kept ( $y$ ) in the tug-of-war version of the game. The right pane shows the correlation of the model with the data. The model predictions (the 3D surface) is plotted next to the experimental data points (blue dots). The model suggests that if the group is contributing, the player is less likely to keep. However, if the player has been successful in total earnings and kept in the previous round, she/he is more likely to continue keeping. The fitted model parameters shown are  $\alpha = 0.0319059$ ,  $\beta = 4.52439$ ,  $\gamma = 1.90551$ .

We can write a general condition for players to contribute to the public good ( $x$ ) by solving for the threshold value analytically:

*Likely to contribute to public good if:*

$$3.35 y \left( \frac{q}{\text{group}(q)} \right) + z \left( \frac{q}{\text{group}(q)} \right) < 1.18$$

The best model obtained for the amount kept ( $y$ ) is shown in Figure 20.5.

For a third time, we see the ratio of the cumulative earnings to the mean of the group arising in the data-driven model. This model suggests that the player is less likely to keep if the mean group contribution is high. However, the player is more likely to



keep if she/he has high cumulative earnings relative to the group and kept in the previous round.

We can write a general condition for players to keep ( $y$ ) by solving for the threshold value analytically:

***Likely to keep fraction if:***

$$0.42 y \left( \frac{q}{\text{group}(q)} \right) - \text{group}(x) > -0.0071$$

The best model found for the fraction invested in competition ( $z$ ) is shown in Figure 20.6.

This model suggest that if the group kept a lot and the player also kept a lot in the previous round, she/he is less lightly to invest in competition in the next round. Additionally, if the group contributed to the public good a lot and the player invested in competition, she/he is more likely to invest in competition again.

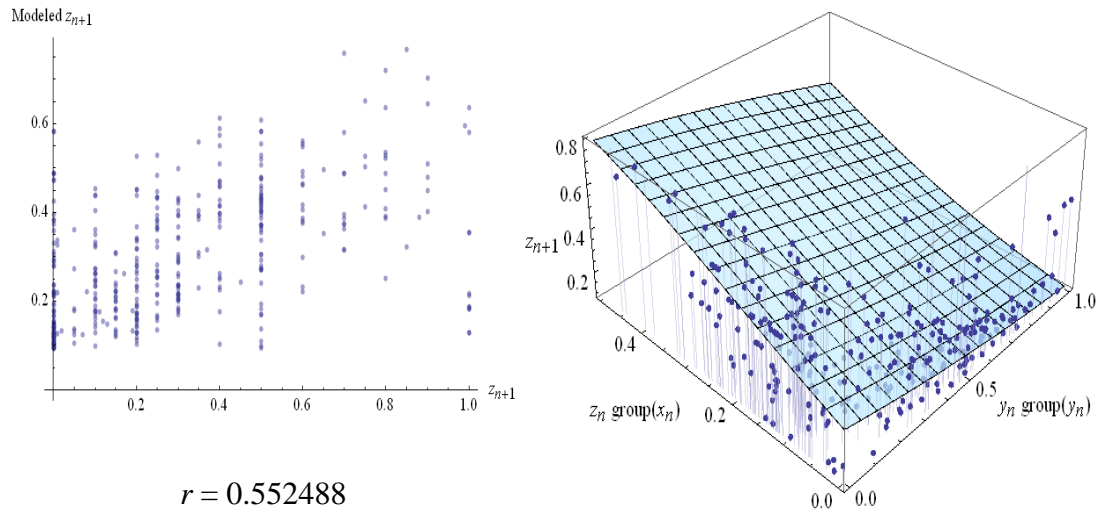
***Likely to compete fraction if:***

$$3.02 z \text{group}(x) - y \text{group}(y) > 0.57$$

We can write a general condition for players to compete ( $z$ ) by solving for the threshold value analytically (above). This translates the model into a more qualitative condition based on the logistic function.

Next, let's again assume that these models capture the mean group behavior and look at the fitness landscape of an individual player against the field. As before, we substitute the modeled  $x_{n+1}$  in for the mean  $\text{group}(x)$ ,  $y_{n+1}$  for  $\text{group}(y)$ , etc. in the fitness function and assume the player's cumulative earnings are equal to the group ( $q$

**Model:**  $z_{n+1} = \text{logistic}(-\alpha - \beta y_n \text{group}(y_n) + \gamma z_n \text{group}(x_n))$



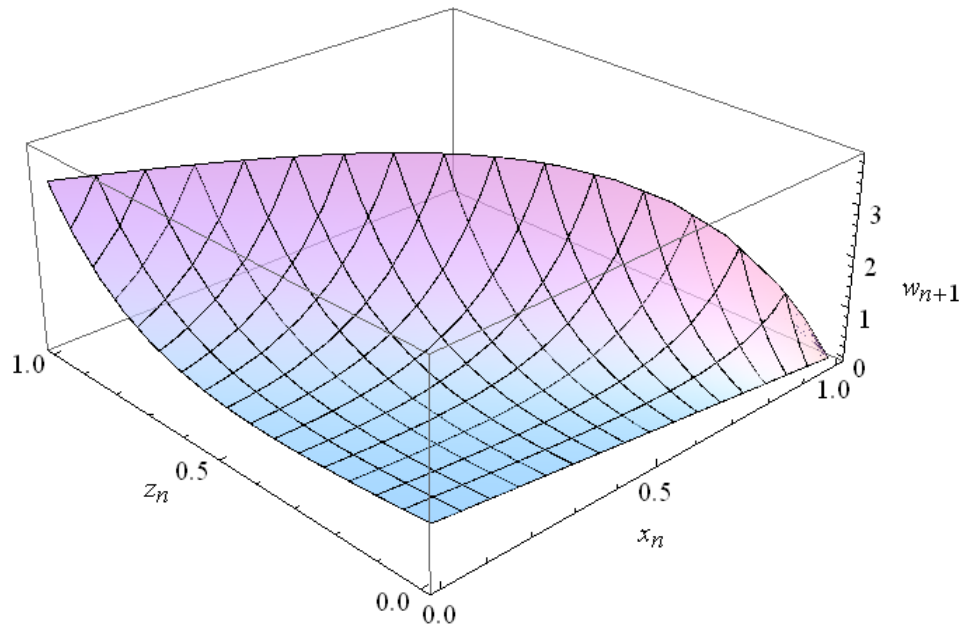
**Figure 20.6.** The model obtained for an individual’s fraction invested in competition ( $z$ ) in the tug-of-war version of the game. The left pane shows the linear correlation of the model predictions with the data. The right pane shows the model predictions (the 3D surface) next to the experimental data. The model suggests that if the group is keeping and the player kept on the previous round, she/he is less likely to compete. Secondly, if the group is contributing and the player competed in the previous round, she/he is more likely to compete again. The fitted model parameters shown are  $\alpha = 0.81949$ ,  $\beta = 1.44882$ ,  $\gamma = 4.38114$ .

$\approx \text{group}(q)$ ). The fitness function then becomes:

$$w = 1 - x - z + z \frac{2 + 7.36 e^{-0.4829 x - 1.9433 z}}{1 + 19.952 e^{-4.6188 x - 3.2425 z}}$$

Unfortunately, I could not analyze this effective fitness analytically – Mathematica was unable to solve for the fixed points. However we can try to analyze it graphically.

Figure 20.7 shows the effective fitness landscape of an individual playing against the field, assuming the group plays according to the  $x_{n+1}$  and  $y_{n+1}$  models. Unlike in the normal game model, we can’t plot all group behaviors at once because there are too many variables to plot. This surface corresponds to when the group played 50% in contribution ( $x$ ) and 50% competition ( $z$ ) in the previous round.



**Figure 20.7.** The effective fitness landscape for an individual playing against the field, assuming the group plays according to the  $x_{n+1}$  and  $y_{n+1}$  models, and in the previous round the group played 50% in contribution ( $x_n$ ) and 50% in competition ( $z_n$ ). The optimal behavior for the individual in this circumstance is to play similarly: investing at least 50% in competition and the rest in contribution. The fitness surface predicted by the model looks similar for other group conditions, most with optima at  $z_n = 100\%$ .

Here the optimal behavior for the individual is to invest at least 50% in competition ( $z$ ) and the rest in contribution ( $x$ ). Since this is similar to what the group is doing, it won't change the group values for the next round. So, this represents a partially stable scenario where the model predicts the group will converge to high competition ( $z$ ) and contribution ( $x$ ).

Plotting the surface for different group conditions in the previous round produces similar fitness landscapes. Nearly all have optima near 50-100% investment in competition. So this model predicts high amounts of competition persist through successive tug-of-war public good rounds.

## **Discussion**

I think there are two interesting general aspects of these results. First, the appearance and re-appearance of the ratio of the cumulative earnings term and the group earnings is quite curious. It is unusual for this to appear by chance several times. The overall cumulative success in earnings appears to be a strong predictor of the player's behavior.

In the model results, we saw that this success factor caused players to contribute to the public good less (in the case of the normal game), or to continue previous actions if the player has been successful.

The second interesting feature of these models is that all of the modeled influences on contribute ( $x$ ), kept ( $y$ ), and compete ( $z$ ) all appear to be influences that would improve the player's earnings; For example, the player being more likely to contribute if the group is contributing more on average.

This suggests that players are behaving rationally, and are goal oriented toward improving their earnings in response to the group and their previous actions.

## **Conclusions**

In conclusion, this chapter used an automated search to extract several empirical models of player behavior in the Public Goods Game. The modeling algorithm identified a common feature that suggests players may change parts of their behavior based on their cumulative success. The models also suggest that players decide their next behavior as a function of the group's behavior from the previous round.

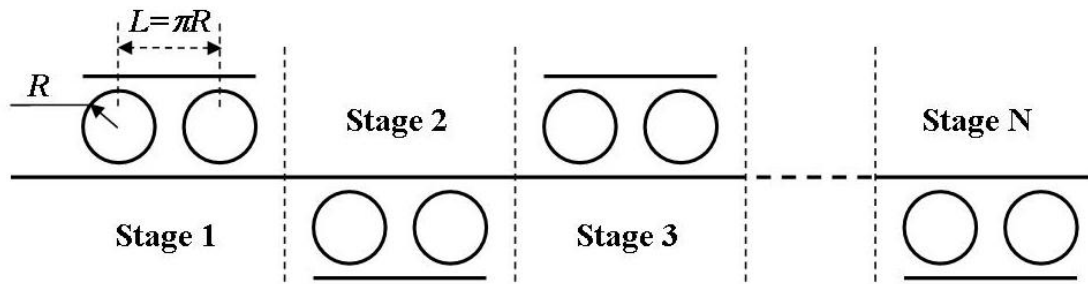
**Summary**

The main challenge in using high Q resonators is their high sensitivity to fabrication errors that affect all the parameters of the resonator. Here we show that solely by choosing carefully the degree of all-optical tuning of the resonators using an evolutionary algorithm, a drastically distorted transmission function can be restored. Results show the ability to combine a small number of devices in series to form arbitrary transmission filters and recover the transmission of damaged devices by re-adjusting dynamic parameters. Finally, we describe a similar approach to reverse-engineer the precise structure and parameters of an unknown optical device directly from observational data.

**Introduction**

High Q ring resonators have been shown to enable novel functionalities on chip (Xu and Lipson 2006; Xu, Sandhu et al. 2006; Xu and Lipson 2007), however the main challenge in using high Q resonators is their high sensitivity to fabrication errors. These errors affect the dimensions of all the parameters of the resonator such as its radius, its quality factor Q and its resonance wavelength. As a results complex transmission functions, that are obtained by coupling such resonators are then drastically distorted.

Here we show that solely by choosing carefully the degree of tuning of the resonators, i.e., its resonance wavelength, a drastically distorted transmission function (due to variations in all of its geometrical parameters) can be recovered. The resonance wavelength of ring resonators can be tuned in silicon using all-optical modulation of the effective index induced due to two photon absorption, as recently demonstrated in



**Figure 21.1. Ring-resonator device structure. Each component contributes to the final transmission.**

(Almeida, Barrios et al. 2004) and (Almeida, Barrios et al. 2004). Here the degree of tuning of each ring's resonances is determined by applying an optimization algorithm to the system.

The structure analyzed is shown in Figure 21.1. Each stage of the device is a double-ring resonator. It consists of a pair of silicon ring resonators coupled to a pair of parallel silicon strip waveguides. The double-ring resonator has a sharp transmission line (EIT-like mode) resulting from a mode due to the interference between the two ring resonators (Xu, Sandhu et al. 2006). By tuning the resonance wavelength of each resonator in the pair, one can control the effective-Q of this mode providing a high degree of freedom to tailor the spectrum. The structure of each stage of the device is defined by the parameters listed in Table 21.1. The range of each parameter corresponds to the one that is typically achieved experimentally in such structures (Xu, Shakya et al. 2006):

**Table 21.1. Range of parameters describing each doubled ring resonator. The parameters controlled externally by all-optical effects are  $\phi_1$  and  $\phi_2$ .**

Variable	Restricted Range
$\kappa$	0.0 – 1.0
$\alpha$	0.0 – 0.005
$\Phi_0$	0 - $2\pi$
$\Phi_1$	0 - $2\pi$
$\Phi_2$	0 - $2\pi$
R	3000 – 10000 nm

$R$  is the nominal radius of each ring resonator.  $\kappa$  is the field coupling coefficient between the waveguide and each ring resonator. It is defined as the amplitude of electric field coupled into the ring divided by the amplitude of electric field in the input waveguide. Its value is between 0 and 1.  $\alpha$  is the field loss per round in the ring. Due to the optical scattering, the amplitude of electrical field drops to  $e^{-\alpha}$  of the original amplitude after it pass one round in the ring. For example  $\alpha = 0.003$  for a ring resonator with radius of 5 microns corresponds to an intrinsic Q of 91,000 for each ring resonator.  $\phi_0$  describes the deviation of the real distance between the two rings from the nominal distance  $\pi R$ , as defined in Figure 21.1.  $\phi_1$  and  $\phi_2$  are defined as the phase shift per round in a ring with radius  $R$  for the light at the wavelength of  $\lambda_1$  and  $\lambda_2$ , respectively. Since the resonant wavelength of each ring resonator depends on the effective index of the ring,  $\phi_1$  and  $\phi_2$  can be tuned at real time using nonlinear effects

**Table 21.2. The parameters for the ring device with transmission function shown in Figure 2 as the dotted line. We simulate manufacturing errors on this device by adding 10% random errors to all parameters. The damaged transmission function is shown in Figure 2 as a dashed line.**

Variable	Device #1	Device #2	Device #3
$\kappa$	0.5785	0.6765	0.6807
$\alpha$	0.0003253	0.002216	1.165e-5
$\Phi_0$	3.9812	5.982	2.011
$\Phi_1$	5.639	2.2582	4.0312
$\Phi_2$	5.482	5.481	4.647
R	8731	7217	8628

in silicon from 0 to  $2\pi$  using for example low power external beams that are incident on the rings for injecting free carriers.

The transmission response of such structure depends strongly on the parameters in Table 21.1. An example of such a spectrum is shown in Figure 21.7 (dotted line) for three rings with parameters listed in Table 21.2. In this chapter, we simulate manufacturing errors by randomly off setting all parameters of each device by 10% of their range. As seen Figure 21.7 (dashed line), such small errors to all devices can drastically change the filter transmission.

In order to compensate for the variation in all the parameters we use an evolutionary algorithm to determine the tuning of only the parameters  $\Phi_1$  and  $\Phi_2$ , in each ring (the



ones that can indeed be controlled externally by injecting carriers using for example two photon absorption induced by another beam of light incident on the ring (Almeida, Barrios et al. 2004; Almeida, Barrios et al. 2004)) that lead to a transmission spectrum that is very close to the original one. To measure the performance of each candidate filter (Ferreira 2002) we define the degree of variation between the original transmission spectrum and the deformed filter by the mean-squared-error:

$$MSE = \sum_i (T(\lambda_i)' - T(\lambda_i))^2$$

where the summation is over a set of training data (a range of wavelengths),  $T(\lambda_i)'$  is the transmission of the damaged filter, and  $T(\lambda_i)$  is the transmission of the designed filter. Solutions with low error (high fitness) are selected (Crow and Kimura 1979) to survive in the population or and to generate new solutions while high error solutions are rejected or replaced. The best candidate solution in the population is tracked over each generation to measure the algorithm's progress. Eventually, the performance of the best solution plateaus after several iterations and the solution is returned. Candidate filter solutions are encoded as a list of parameters for each device (up to 10) of the ring series. New solutions are formed by crossing two low-error solutions in the current population to randomly recombine their parameters. New solutions are also formed by mutation – randomizing some parameters in the filter.

The qualitative box-filter behavior (shown in Figure 21.7 as solid line) is fully restored in the recovery stage by readjusting solely the  $\Phi_1$  and  $\Phi_2$  parameters. Note that some precision is lost. This result however shows that the dynamic parameters  $\Phi_1$  and  $\Phi_2$  alone are powerful enough to restore significant defects. Figure 21.8 describes the remaining error (i.e., distortion of the transmission function) as a function of number

of generation for which the algorithm is run. One can see that after  $10^3$  generations, which corresponds to approximately 5 min of computational efforts on a 3GHz machine, a very small mean square error is achieved.

### **Filter Design Using Evolutionary Computation**

We used an evolutionary algorithm to design arbitrary optical filters using combinations of rings connected in series. In this section, we provide a brief overview of evolutionary algorithms, specifics on our implementation, and results designing a low-pass filter, box-filter, band-pass filter, and ramp filter.

#### ***Evolutionary Algorithm Overview***

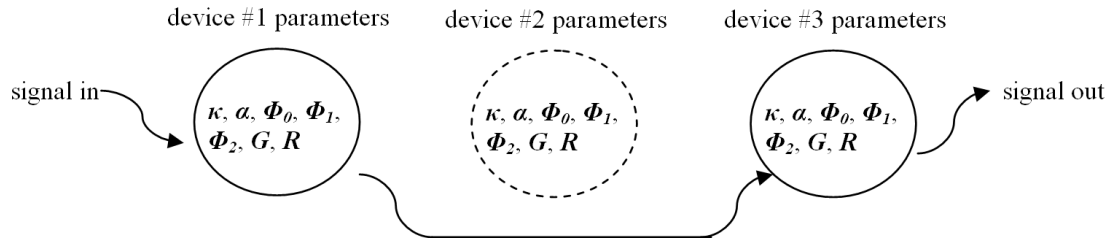
See the description in the section “Evolutionary Computation” on page 3.

#### ***Filter Encoding***

The transmission function of the ring device described in above is accurately described by a 7-tuple list of variables that can be chosen at design time for manufacturing. Certain variables have range restrictions, summarized in Table 21.1, to ensure that the devices can be manufactured and that the transmission formula holds.

Our goal is to combine multiple ring devices in series in order to realize some arbitrary desired transmission function. Therefore, our encoding is contains an ordered list of parameters for each device in the series.

In our experiments, we limit the number of devices used per filter solution to ten. The encoding for each device is given one additional binary variable to indicate if the device is included in the series. If the flag is set to omit the device, the device parameters still exist in the filter encoding but are shorted out or bypassed and need not be manufactured.



**Figure 21.2.. Example filter encoding with a maximum of three devices. Each device consists of seven independent parameters and a flag to include or omit the device from the final series. In this figure, device #2 is flagged as omitted.**

The mutation evolutionary operation for this encoding can randomize a parameter (e.g. set kappa in device #1 to a random value in its allowed range) or randomly include or exclude the device from the series.

The crossover evolutionary operation we use for this encoding is called single-point crossover. One device in the series is picked as the crossover point. The child then inherits the devices before this point from its first parent, and the remaining from its second parent.

### ***Fitness Objective***

Our first experiment is to evolve the parameters of the ring devices in the encoding such that the combined transmission matches some desired function (e.g. a low-pass filter). After picking a desired transmission function, we generate training data of 1000 points over the 1544-1550nm wavelengths.

Our objective is to have the candidate solution filters match the training data as closely as possible. Therefore, we define the fitness objective as the sum-of-squares error from the training data. By convention, we negate the squared error to make the fitness a maximizing objective.

$$fitness(filter) = -\sum_i (filter(x_i) - y_i)^2$$

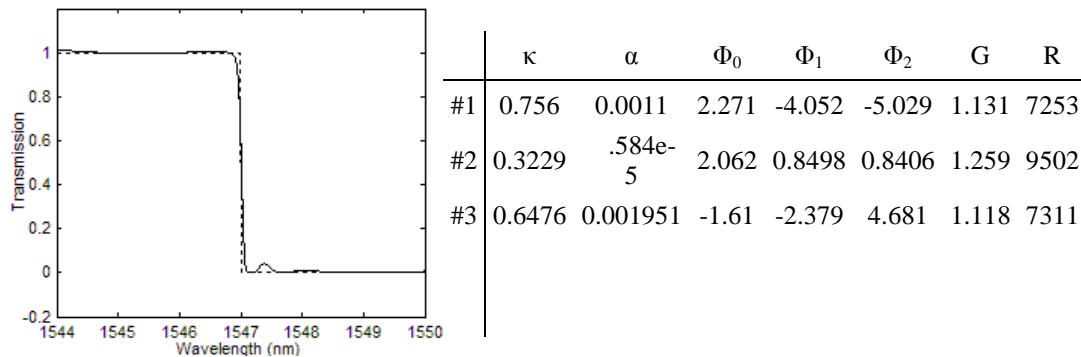
where *filter* is a candidate solution encoding,  $filter(x_i)$  is the output transmission,  $x_i$  is the input transmission, and  $y_i$  is the desired transmission.

### Experimental Setup

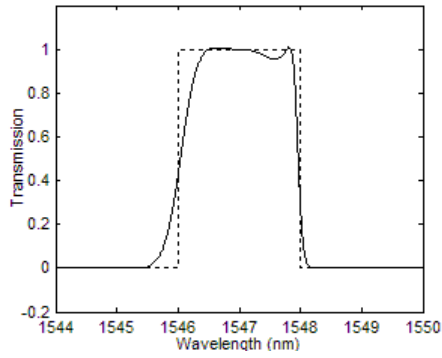
In this experiment we use a basic parametric evolutionary algorithm (Koza 1992) to evolve the filters. We use a population size of 100, crossover probability of 75% and mutation probability of 5%. We use Deterministic Crowding (Mahfoud 1995) to generate successive generations. Solutions are evolved for 10K generations (roughly 10-15 minutes of computing). These are the first settings we tried and are most-likely not optimal, but they turn out to work well.

### Filter Design Results

Here we show we show the best solution found in 10 runs of the algorithm for four common filter types: low-pass, box, band-pass, and ramp. Each run takes approximately 10-15 minutes on a single computer. All runs converged on the general shape of each filter with some approximating the flat regions more closely than others. Here, we show the solutions that gave the tightest fit in Figure 21.3, Figure 21.4, Figure 21.5, and Figure 21.6.

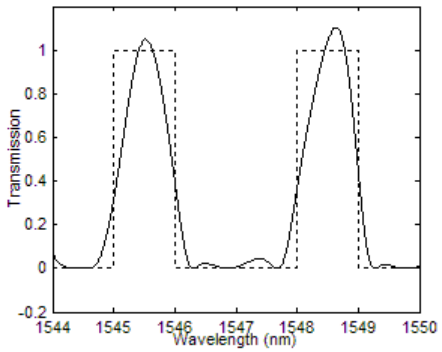


**Figure 21.3. Evolved low-pass filter. The target transmission is shown as a dotted line and the best evolved solution is shown in solid. This solution used three devices in its encoding, shown in the right pane.**



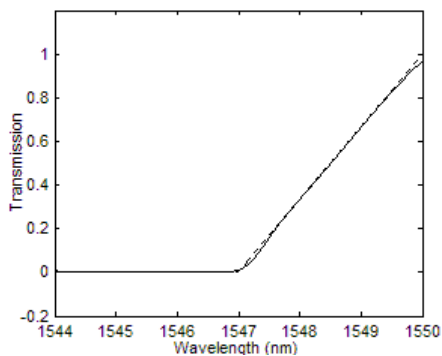
	$\kappa$	$\alpha$	$\Phi_0$	$\Phi_1$	$\Phi_2$	G	R
#1	0.8381	0.001291	4.794	5.923	1.255	1.1	8940
#2	0.3981	0.001429	1.385	2.361	-1.48	1.315	5070
#3	0.6987	0.002343	-3.96	0.977	-4.26	1.379	8580
#4	0.6304	0.000572	1.31	0.716	-5.7	1.391	5908
#5	0.7856	0.003336	-3.13	-0.35	-1.54	1.132	96 2

**Figure 21.4. Evolved box filter.** The target is shown as a dotted line, the best evolved solution is shown in solid. This solution used five devices in its encoding, shown in the right pane.



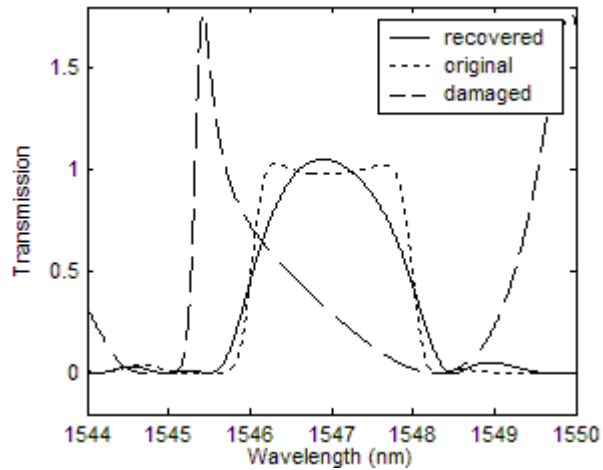
	$\kappa$	$\alpha$	$\Phi_0$	$\Phi_1$	$\Phi_2$	G	R
#1	0.4413	0.00239	-3.373	-5.627	1.591	1.122	9 45
#2	0.443	0.004919	-1.945	4.56	-4.715	1.073	9482
#3	0.762	0.004901	3.232	4.77	5.705	1.253	9390
#4	0.6479	0.001545	0.1748	2.714	-2.263	1.145	6894

**Figure 21.5. Evolved band-pass filter.** The target transmission is shown as a dotted line, the best solution is shown in solid. This solution used four devices in its encoding, shown in the right pane.



	$\kappa$	$\alpha$	$\Phi_0$	$\Phi_1$	$\Phi_2$	G	R
#1	0.7514	0.000583	-2.387	3.117	-4	1.216	5562
#2	0. 817	0.0006682	-5.122	5.754	-4.701	1.338	3349

**Figure 21.6. Evolved ramp filter.** The target transmission is shown as a dotted line, the best evolved solution is shown in solid. This solution used two devices in its encoding, shown in the right pane.



**Figure 21.7. A damaged five-device filter. The  $\kappa$ ,  $\alpha$ ,  $\Phi_0$ ,  $R$ , and  $G$  parameters are offset by 10% random manufacturing error. The qualitative box-filter transmission function has been restored however some precision is still lost.**

In these experiments, solutions converged to the qualitative shape of the target transmissions very quickly in less than 1000 generations. Remaining generations produced successive approximations to the “flat” regions and discontinuous regions of the target filter.

### **Error and Damage Recovery Results**

Small damage or manufacturing errors can cause deviations from the intended functionality of optical devices.

#### *Error and Damage Effects*

In this experiment we damaged the  $\kappa$ ,  $\alpha$ , and  $\Phi_0$  parameters of each device in an evolved filter by adding a random 10% offset, see Figure 21.7. An obvious consequence of even small error is that it can propagate large error to successive components leading to meta-stable or unpredictable states.

Fortunately the  $\Phi_1$  and  $\Phi_2$  phases on each device can be re-tuned dynamically. Using the same evolutionary algorithm as before, we can find the optimal  $\Phi_1$  and  $\Phi_2$  settings

on the damaged devices to attempt to restore the series to its original transmission.

The evolutionary algorithm is identical to the previous experiment except now the parameters  $\kappa$ ,  $\alpha$ ,  $\Phi_0$ ,  $G$ , and  $R$  have been fixed assuming they have been fabricated. The previous algorithm is modified by hard-coding the fixed parameters to their damaged values and only evolving the  $\Phi_1$  and  $\Phi_2$  parameters on each device.

### *Damage Recovery Results*

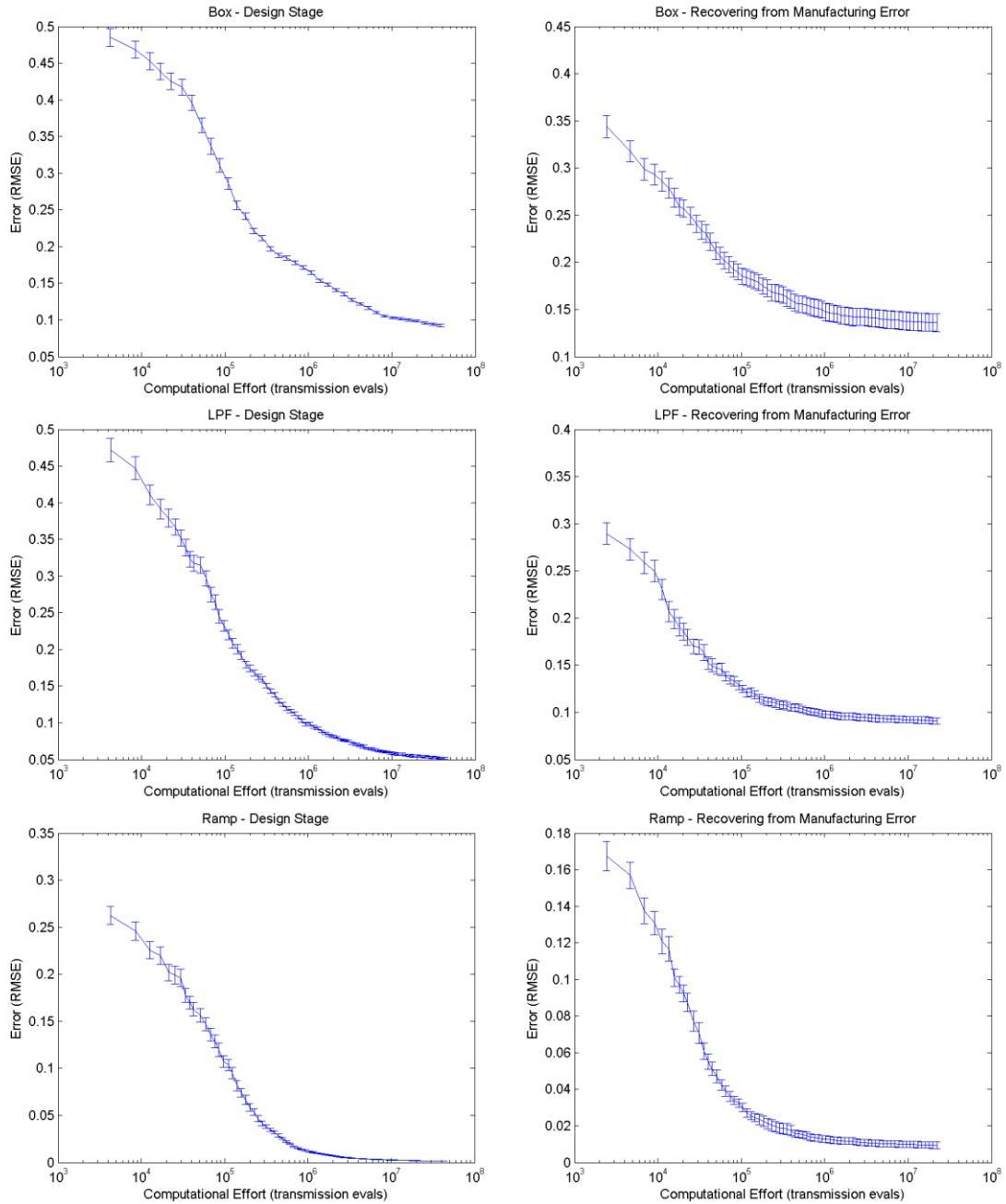
Results in this experiment show the evolutionary algorithm both finding the desired transmission filter and recovering the designed functionality after all parameters have suffered 10% manufacturing error or damage.

The experiment has three steps:

1. Evolve an optimal filter for the desired transmission function
2. The designed filter undergoes 10% error on all parameters
3. Evolve the tunable  $\Phi_1$  and  $\Phi_2$  parameters to recover the designed transmission

The 10% random error to all device parameters significantly changes the filter transmission, shown in Figure 21.7. The qualitative box-filter behavior is fully restored in the recovery stage by readjusting the  $\Phi_1$  and  $\Phi_2$  parameters, however, some precision is lost that cannot be recovered, shown in Figure 21.8.

It is possible that larger errors and damage could be unrecoverable. Results in Figure 21.8 show however that the dynamic parameters  $\Phi_1$  and  $\Phi_2$  alone are powerful enough to repair substantial defects.



**Figure 21.8. Recovering a damaged device – box, LPF, and ramp filters. The left pane shows the error of the best filter being evolved before fabrication. 10% random error is then added to all fixed parameters on all devices. The right pane shows the best filter being evolved to recover from manufactured errors. Errorbars show the standard error.**

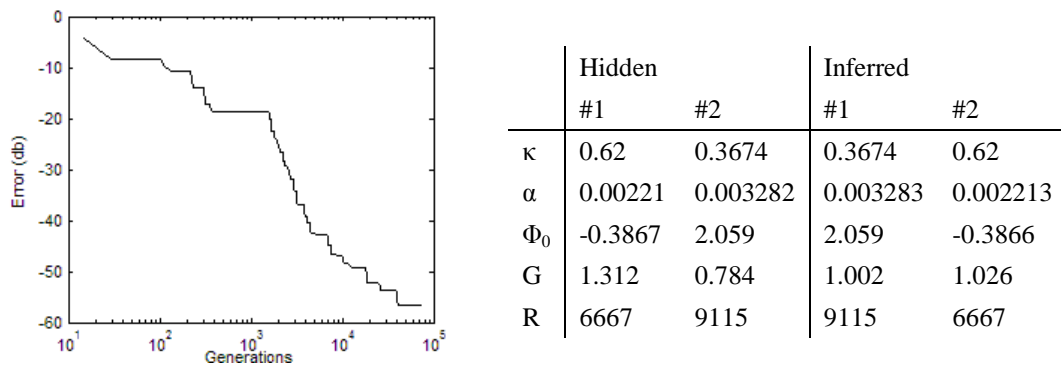


## Inferring Parameters of a Built Device

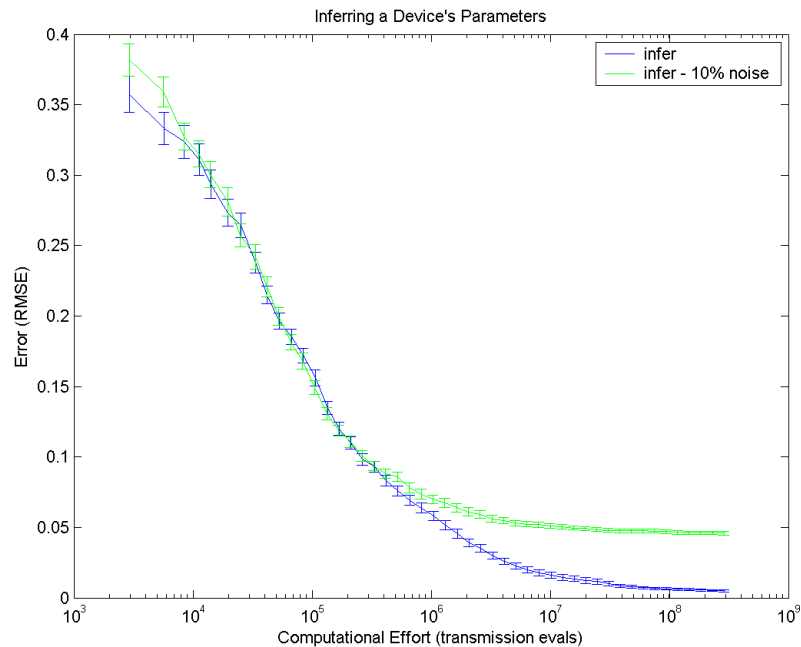
In this section we investigate the related problem of modeling an existing optical system. The objective is now to reverse-engineer the parameters of a given physical device. Due to variation in manufacturing, physical devices may not match designed behavior exactly. Using the same evolutionary algorithm in the first experiment, we can search for the exact parameters that were realized in the manufactured device.

The procedure is identical to the filter design procedure in the first experiment except, rather than evolving device parameters to match some theoretical desired transmission, we can evolve parameters to match the transmission measured experimentally from a physical device. Instead of generating training data synthetically, we can generate training data by measuring the transmission of the physical device and use the same fitness objective as before.

Since the training data now comes from a real device, there must exist a set of parameters that will match the measured training data exactly, assuming our device model is sufficient. Therefore, we can continue evolution until solutions converge



**Figure 21.9. Inferring the physical parameters of a 2-device filter. The inferred model matches the hidden system to within very low error. Note that the order of devices and the gain levels of each individual device cannot be determined due to algebraic properties of multiplying the transmission of each device. The total gain is inferred correctly however ( $1.312 \cdot 0.784 = 1.002 \cdot 1.026 = 1.0028$ ).**



**Figure 21.10. Reverse-engineering random 4-device filters give precise transmission measurements (blue) and noisy transmission measurements (green). Error bars show the standard error.**

precisely on the measured data. We can then use the converged solution to infer the exact parameters that were manufactured directly from observational data alone.

In this experiment, inferring the device parameters of an unknown filter takes significantly more computation than designing a filter since we must evolve models to convergence (very low error). This experiment ran 75K generations, or roughly one hour on a single computer.

The hidden system is inferred correctly, shown in Figure 21.9. Note however that the order of devices is switched and the gains of each individual device do not match. This is a result of the multiplicative properties of a signal transmitting between devices. The total gain of the filter is inferred correctly – the product of the gain of each device in each filter is identical.

Figure 21.10 shows the results from inferring parameters of random 4-device filters. Given precise measurements from the physical device, the algorithm finds the filter model reliably with low error. If only noisy measurements are available from the physical device, the algorithm finds the device consistently at the same rate, but error measured asymptotes sooner because the noise cannot be modeled exactly.

### **Conclusions**

In conclusion we have shown the ability to overcome manufacturing variations and recover the intended functionality by determining the parameters that can be controlled externally using all-optical effects

We have shown the ability to find combinations of optical devices and parameters to design arbitrary transmission filters. If a device is damaged after manufacturing, we have shown that the dynamical parameters can be re-adjusted with the same algorithm to recover the intended functionality. Finally, we propose using this approach in the future to reverse-engineer the parameters of an unknown manufactured device directly from observational data.

## CONTRIBUTIONS

### **Primary Contributions**

#### *Chapter 4*

- Introduced a new algorithm based on coevolution and approximating fitness calculations to reduce computational cost
- Demonstrated substantial improvements in performance over previous and alternative methods

#### *Chapter 5*

- Introduced a new algorithm based on comparing and predicting ranks of solutions to accelerate performance

#### *Chapter 6*

- Introduced a new multi-objective evolutionary algorithm that optimizes solution genotypic age
- Demonstrated improvement over previous population-structure methods
- Introduced an algorithm based on using multiple secondary objectives
- Analyzed the impact of different combinations of secondary objectives: error, age, complexity, and novelty

#### *Chapter 7*

- Introduced and compared several new methods for reusing prior models or knowledge in an evolutionary search

#### *Chapter 8*

- Introduced a new algorithm for extracting meaningful model building-blocks based on intersecting modeling results from multiple systems

### ***Chapter 9***

- Described new techniques for identifying ODE models from experimental data
- Demonstrated identifying a variety of dynamical systems

### ***Chapter 10***

- Introduced a new criterion for identifying nontrivial implicit equations
- Demonstrated identifying a variety of surfaces and invariant manifolds

### ***Chapter 11***

- Introduced a new method to infer physical laws from raw experimental data based on identifying invariant quantities in time-series data
- Demonstrated detecting conserved quantities from motion tracking data

### ***Chapter 12***

- Introduced a new method to represent and evolve noise sources and model stochastic elements explicitly in a symbolic model
- Introduced a new fitness metric to identify models with the simplest noise envelope that enclosed the experimental data

### ***Chapter 13***

- Described new techniques for evolving stochastic reaction models
- Introduced a new fitness metric to identify a maximum likelihood stochastic model from experimental data, even when likelihood estimates are unavailable or inaccurate
- Demonstrated the approach on the Lotka-Volterra system using sparse data with large time gaps between measurements

### ***Chapter 14***

- Introduced a new encoding for symbolic expressions based on an acyclic graph equation encoding, and compared with ordinary tree encodings

### ***Chapter 15***

- Introduced a new method for relating an automatically inferred model to a prior, well-understood model based on identifying a mapping between model parameters
- Introduced a new technique for evolving symbolic models that can use multiple different coefficients values to model different experiments on the same system (e.g. time series of different cells)
- Identified a new and simpler dynamical model of bacillus competence
- Identified a new conserved quantity in bacillus competence, found to be related to the cell's competence duration

### ***Chapter 16***

- Introduced a new method for identifying a fully-parameterized model from an automatically inferred model with bulk coefficients

### ***Chapter 17***

- Introduced new techniques for modeling and designing experiments to analyze metabolic networks
- Demonstrated inferring a yeast metabolism model from noisy time-series data, the largest and most nonlinear dynamical automatically identified ODE system at the time
- Demonstrated correcting a manually-derived model and a closely related model from another system using experimental data

- Demonstrated substantial improvements in numerical prediction over other regression methods

### ***Chapter 18***

- Introduced a new method for testing the importance of individual building-blocks based on analyzing cross-validation curves and functional data analysis
- Demonstrated modeling of motion-captured insect wing dynamics

### ***Chapter 19***

- Introduced a new technique for modeling and inferring a human user's preference when comparing two artistic drawings
- Demonstrated inferring a user's preference for square and star shaped drawings

### ***Chapter 20***

- Introduced a new technique for modeling a human user's strategy when playing Public Goods Games
- Identified several new qualitative trends in player strategies

### ***Chapter 21***

- Introduced a new algorithm for designing optical filters with arbitrary transmission functions
- Demonstrated tuning optical filters to correct manufacturing errors

## **Contributions of Others**

### ***Chapter 11***

- Analyzed a double pendulum provided by Professor Andy Ruina (Cornell University)
- Used motion tracking system and software provided by Professor John

Guckenheimer (Cornell University)

### ***Chapter 13***

- Professor Gürol Süel (University of Texas Southwestern Medical Center) conceived and suggested the stochastic modeling problem

### ***Chapter 15***

- Professor Gürol Süel (University of Texas Southwestern Medical Center), conceived the project, performed biological experiments, analyzed modeling results, and helped write and edit significant portions of text
- Instructor Tolga Çagatay (University of Texas Southwestern Medical Center) also performed biological experiments, analyzed results, and helped write and edited significant portions of text

### ***Chapter 17***

- John Wikswo (Vanderbilt University) conceived and designed the project
- John Wikswo (Vanderbilt University), Jonathan Hood (CFD Research Corporation), and Abhishek Soni (CFD Research Corporation) developed the forward model
- Ravishankar Vallabhajosyula (CFD Research Corporation) and Jonathan Hood (CFD Research Corporation) interpreted the invariant results
- Ravishankar Vallabhajosyula (CFD Research Corporation), John Wikswo (Vanderbilt University), and Jerry Jenkins (HudsonAlpha Institute) helped write and revise text

### ***Chapter 18***

- Professor Giles Hooker (Cornell University) advised the project
- Atilla Bergou and Gordon Berman (Cornell University) performed



experiments and provided data

### *Chapter 20*

- Professor Kern Reeve (Cornell University) advised the project
- Jessie Barker (Cornell University) conducted experiments, provided data, and help analyze results

### *Chapter 21*

- Professor Michal Lipson (Cornell University) advised the project, wrote text, and analyzed results
- Qianfan Xu (Cornell University) developed the optical device model, helped design the optimization parameters, wrote text, and analyzed results

## REFERENCES

- Abramson, I. S. (1982). "On bandwidth variation in kernel estimates--a square root law." Ann. Statist. **10**(4): 1217-1223.
- Albert, L. A. and D. E. Goldberg (2002). Efficient Discretization Scheduling In Multiple Dimensions. Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers Inc.
- Almeida, V. R., C. A. Barrios, et al. (2004). "All-optical control of light on a silicon chip." Nature **431**(7012): 1081-1084.
- Almeida, V. R., C. A. Barrios, et al. (2004). "All-optical switching on a silicon chip." Opt. Lett. **29**(24): 2867-2869.
- Anderson, P. W. (1972). "More Is Different." Science **177**(4047): 393-396.
- Anderson, P. W. and E. Abrahams (2009). "Machines Fall Short of Revolutionary Science." Science **324**(5934): 1515-1516.
- Arnold, D. V. (2001). Evolution strategies in noisy environments- a survey of existing work. Theoretical aspects of evolutionary computing, Springer-Verlag: 239-250.
- Audet, C., J. J. E. Dennis, et al. (2000). "Surrogate-Model-Based Method For Constrained Optimization." AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.
- Auger, A. and N. Hansen (2005). "A restart CMA evolution strategy with increasing population size." Evolutionary Computation, 2005. The 2005 IEEE Congress on 2: 1769- 1776.
- Augusto, D. A. and H. J. C. Barbosa (2000). Symbolic Regression via Genetic Programming. VI Brazilian Symposium on Neural Networks (SBRN'00), Rio de Janeiro, RJ, Brazil.

- Ball, P. (2009). Physics by numbers. Nature News.
- Bansal, M., V. Belcastro, et al. (2007). "How to infer gene networks from expression profiles." Mol Syst Biol **3**.
- Bansal, M., G. D. Gatta, et al. (2006). "Inference of gene regulatory networks and compound mode of action from time course gene expression profiles." Bioinformatics **22**(7): 815-822.
- Banzhaf, W. and W. B. Langdon (2002). Some considerations on the reason for bloat. Genetic Programming and Evolvable Machines. **3**: 81--91.
- Banzhaf, W. and J. Miller (2004). The Challenge of Complexity. Frontiers of Evolutionary Computation: 243-260.
- Bautu, E., A. Bautu, et al. (2005). Symbolic Regression on Noisy Data with Genetic and Gene Expression Programming. Proceedings of the Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, IEEE Computer Society.
- Beard, D. A., H. Qian, et al. (2004). Stioichiometric foundation of large-scale biochemical system analysis. Modelling in molecular biology. Berlin; New York, Springer: 1-19.
- Bishop, I. D. (1996). "Comparing regression and neural net based approaches to modelling of scenic beauty." Landscape and Urban Planning **34**(2): 125-134.
- Bonabeau, E. (2002). "Agent-based modeling: Methods and techniques for simulating human systems." Proceedings of the National Academy of Sciences of the United States of America **99**(Suppl 3): 7280-7287.
- Bongard, J. and H. Lipson (2007). "Automated reverse engineering of nonlinear dynamical systems." Proceedings of the National Academy of Sciences **104**(24): 9943-9948.
- Bongard, J., V. Zykov, et al. (2006). "Resilient Machines Through Continuous Self-

- Modeling." Science **314**(5802): 1118-1121.
- Bongard, J. C. and H. Lipson (2005). 'Managed challenge' alleviates disengagement in co-evolutionary system identification. Proceedings of the Genetic and Evolutionary Computation Conference, Washington DC, USA, ACM.
- Bongard, J. C. and H. Lipson (2005). "Nonlinear System Identification Using Coevolution of Models and Tests." IEEE Transactions on Evolutionary Computation **9**(4): 361-384.
- Bonneau, R., D. Reiss, et al. (2006). "The Inferelator: an algorithm for learning parsimonious regulatory networks from systems-biology data sets de novo." Genome Biology **7**(5): R36.
- Booker, A. J., J. E. Dennis, Jr., et al. (1999). "A rigorous framework for optimization of expensive functions by surrogates." Structural Optimization **17**(1): 1-13.
- Bucci, A. and J. B. Pollack (2005). On identifying global optima in cooperative coevolution. Proceedings of the Genetic and Evolutionary Computation Conference, Washington DC, USA, ACM.
- Çagatay, T., M. Turcotte, et al. (2009). "Architecture-Dependent Noise Discriminates Functionally Analogous Differentiation Circuits." **139**(3): 512-522.
- Cao, Y., D. T. Gillespie, et al. (2005). "Avoiding negative populations in explicit Poisson tau-leaping." J Chem Phys **123**(5): 054104.
- Carl Edward, R. (1997). Evaluation of gaussian processes and other methods for non-linear regression, University of Toronto: 127.
- Casey, S. G., C. W. Bill, et al. (2008). Using expert knowledge in initialization for genome-wide analysis of epistasis using genetic programming. Proceedings of the 10th annual conference on Genetic and evolutionary computation. Atlanta, GA, USA, ACM.
- Chen, J.-H., D. E. Goldberg, et al. (2002). Fitness Inheritance In Multi-objective

- Optimization. Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers Inc.
- Christian, G., P. Marc, et al. (2003). A Robust Master-Slave Distribution Architecture for Evolutionary Computations. Genetic and Evolutionary Computation Conference Late Breaking Papers. R. Bart. Chicago, USA: 80--87.
- Clery, D. and D. Voss (2005). "All for One and One for All." Science **308**(5723): 809.
- Cleveland, W. S. and S. J. Devlin (1988). "Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting." Journal of the American Statistical Association **83**: 596-610.
- Cliff, D. and G. F. Miller (1995). Tracking the Red Queen: Measurements of Adaptive Progress in Co-Evolutionary Simulations. Proceedings of the Third European Conference on Advances in Artificial Life, Springer-Verlag.
- Conor, R. (1996). Reducing Premature Convergence in Evolutionary Algorithms. Ireland, University College, Cork.
- Crampin, E. J., S. Schnell, et al. (2004). "Mathematical and computational techniques to deduce complex biochemical reaction mechanisms." Progress in Biophysics & Molecular Biology **86**(1): 77-112.
- Crow, J. F. and M. Kimura (1979). "Efficiency of truncation selection." Proceedings of the National Academy of Sciences of the United States of America **76**(1): 396-399.
- Cybenko, G. (1992). "Approximation by superpositions of a sigmoidal function." Mathematics of Control, Signals, and Systems (MCSS) **5**(4): 455-455.
- Cyril, F. and B. Alberto (2007). Symbolic Regression of Discontinuous and Multivariate Functions by Hyper-Volume Error Separation (HVES). 2007 IEEE Congress on Evolutionary Computation. S. Dipti and W. Lipo. Singapore, IEEE Press IEEE Computational Intelligence Society.

- David, H. W. (1997). "On bias plus variance." Neural Comput. **9**(6): 1211-1243.
- Dawkins, R. (1996). The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe without Design, W.W. Norton & Co.
- De Falco, I., E. Tarantino, et al. (2002). Unsupervised spectral pattern recognition for multispectral images by means of a genetic programming approach. Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on.
- De Jong, E. D. and J. B. Pollack (2004). "Ideal Evaluation from Coevolution." Evolutionary Computation **12**(2): 159-192.
- di Bernardo, D., M. J. Thompson, et al. (2005). "Chemogenomic profiling on a genome-wide scale using reverse-engineered gene networks." Nat Biotech **23**(3): 377-383.
- Dolin, B., F. H. B. III, et al. (2002). Co-evolving an effective fitness sample: experiments in symbolic regression and distributed robot control. Proceedings of the 2002 ACM symposium on Applied computing, Madrid, Spain, ACM.
- Domingos, P. (2000). A unified bias-variance decomposition and its applications. In Proc. 17th International Conf. on Machine Learning.
- Doob, J. L. (1945). "Markoff chains--denumerable case " Trans. Amer. Math. Soc. **58**: 455-473.
- Duffy, J. and J. Engle-Warnick (2002). "Using Symbolic Regression to Infer Strategies from Experimental Data." Evolutionary Computation in Economics and Finance **100**(4): 61--84.
- Edwin, D. and B. P. Jordan (2003). Multi-Objective Methods for Tree Size Control. Genetic Programming and Evolvable Machines. **4**: 211--233.
- Eggermont, J. and J. I. v. Hemert (2000). Stepwise Adaptation of Weights for Symbolic Regression with Genetic Programming. Proceedings of the Twelveth

- Belgium/Netherlands Conference on Artificial Intelligence (BNAIC'00), De Efteling, Kaatsheuvel, Holland, BNVKI, Dutch and the Belgian AI Association.
- Ekárt, A. and S. Z. Németh (2001). "Selection Based on the Pareto Nondomination Criterion for Controlling Code Growth in Genetic Programming." Genetic Programming and Evolvable Machines **2**(1): 61-73.
- Elena, B., B. Andrei, et al. (2005). Symbolic Regression on Noisy Data with Genetic and Gene Expression Programming. Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'05): 321--324.
- Evans, J. and A. Rzhetsky "Machine Science." Science **329**(5990): 399-400.
- Fell, D. A. (1992). "Metabolic Control Analysis - A Survey of Its Theoretical and Experimental Development." Biochem.J. **286**: 313-330.
- Ferreira, C. (2002). Function Finding and the Creation of Numerical Constants in Gene Expression Programming. 7th Online World Conference on Soft Computing in Industrial Applications.
- Ficici, S. G. (2004). Solution Concepts in Coevolutionary Algorithms. Computer Science, Brandeis University.
- Ficici, S. G. and J. B. Pollack (2001). Pareto Optimality in Coevolutionary Learning. Proceedings of the 6th European Conference on Advances in Artificial Life, Springer-Verlag.
- Fogel, L. J., A. J. Owens, et al. (1966). Artificial Intelligence through Simulated Evolution, John Wiley.
- Fonseca, C. and P. Fleming (1993). Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. Genetic Algorithms: Proceedings of the Fifth International Conference, Morgan

Kaufmann.

- Fonseca, C. M. and P. J. Fleming (1995). "An Overview of Evolutionary Algorithms in Multiobjective Optimization." Evolutionary Computation **3**(1): 1-16.
- Forrest, S. (1993). "Genetic algorithms: principles of natural selection applied to computation." Science **261**(5123): 872-878.
- Francisco, F., S. Giandomenico, et al. (2005). Parallel Genetic Programming. Parallel Metaheuristics. A. Enrique. Hoboken, New Jersey, USA, Wiley-Interscience: 127--153.
- Francisco, F., T. Marco, et al. (2003). An Empirical Study of Multipopulation Genetic Programming. Genetic Programming and Evolvable Machines. **4**: 21--51.
- Franz, R. (2006). Representations for genetic and evolutionary algorithms. Springer Verlag.
- Gadkar, K. G., J. Varner, et al. (2005). "Model identification of signal transduction networks from data using a state regulator problem." Systems Biology **2**(1): 17-30.
- Gardner, T. S., D. di Bernardo, et al. (2003). "Inferring Genetic Networks and Identifying Compound Mode of Action via Expression Profiling." Science **301**(5629): 102-105.
- Gianfelici, F. (2010). "Machine Science: Truly Machine-Aided Science." Science **330**(6002): 317.
- Gillespie, D. T. (1977). "Exact Stochastic Simulation of Coupled Chemical Reactions." The Journal of Physical Chemistry **81**(25): 2340-2361.
- Gillespie, D. T. (2007). "Stochastic simulation of chemical kinetics." Annu Rev Phys Chem **58**: 35-55.
- Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Longman Publishing Co., Inc.



- Goldbeter, A. (1996). Biochemical Oscillations and Cellular Rhythms : the Molecular Bases of Periodic and Chaotic Behaviour. Cambridge, New York, Cambridge University Press.
- Goldbeter, A. and R. Lefever (1972). "Dissipative Structures for an Allosteric Model: Application to Glycolytic Oscillations." Biophys. J. **12**(10): 1302-1315.
- Grefenstette, J. J. (1988). "Credit assignment in rule discovery systems based on genetic algorithms." Machine Learning **3**(2): 225-245.
- Greg, W. and F. Eric (2002). Motion Tracking: No Silver Bullet, but a Respectable Arsenal. **22**: 24-38.
- Gregory, P., R. Denis, et al. (2003). Exploring Overfitting in Genetic Programming. Evolution Artificielle, 6th International Conference. L. Pierre, C. Pierre, F. Cyril, L. Evelyne and S. Marc. Marseilles, France, Springer. **2936**: 267--277.
- Grünwald, P. (2000). "Model selection based on minimum description length." J. Math. Psychol. **44**(1): 133-152.
- Hahn, J., A. Luttinger, et al. (1996). "Regulatory inputs for the synthesis of ComK, the competence transcription factor of Bacillus subtilis." Mol Microbiol **21**(4): 763-75.
- Hanc, J., S. Tuleja, et al. (2004). "Symmetries and conservation laws: Consequences of Noether's theorem." American Journal of Physics **72**(4): 428-435.
- Hardin, G. (1968). "The Tragedy of the Commons." Science **162**(3859): 1243-1248.
- Haufe, C., K. C. Elliott, et al. (2010). "Machine Science: What's Missing." Science **330**(6002): 317-318.
- Heinrich, R., S. M. Rapoport, et al. (1977). "Metabolic-Regulation and Mathematical-Models." Prog. Biophys. Mol. Biol. **32**: 1-82.
- Hermann, N. (1995). "On the Probabilistic Interpretation of Neural Network Classifiers and Discriminative Training Criteria." IEEE Trans. Pattern Anal.

- Mach. Intell. **17**(2): 107-119.
- Hetland, M. L. and P. Sætrum (2005). "Evolutionary Rule Mining in Time Series Databases." Machine Learning **58**(2): 107-125.
- Hideaki, S. and S. Shigeru (2007). "A Method for Selecting the Bin Size of a Time Histogram." Neural Comput. **19**(6): 1503-1527.
- Higgins, J. (1964). "A Chemical Mechanism for Oscillation of Glycolytic Intermediates in Yeast Cells." Proceedings of the National Academy of Sciences **51**(6): 989-994.
- Hillis, W. D. (1991). Co-evolving parasites improve simulated evolution as an optimization procedure. Emergent computation, MIT Press: 228-234.
- Hoai, N. X., R. I. McKay, et al. (2002). Solving the Symbolic Regression Problem with Tree-Adjunct Grammar Guided Genetic Programming: The Comparative Results. Proceedings of the 2002 Congress on Evolutionary Computation, IEEE Neural Network Council (NNC), Institution of Electrical Engineers (IEE), Evolutionary Programming Society (EPS).
- Holland, J. (1975). Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence, University of Michigan Press.
- Holland, J. H. (2000). "Building Blocks, Cohort Genetic Algorithms, and Hyperplane-Defined Functions." Evolutionary Computation **8**(4): 373-391.
- Hollnagel, E. (1993). Human Reliability Analysis: Context and Control. New York, NY, Academic Press, Inc.
- Hornby, G. S. (2006). ALPS: the age-layered population structure for reducing the problem of premature convergence. GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation. K. Maarten, C. Mike, A. Dirk et al. Seattle, Washington, USA, ACM Press New York, NY,

- 10286-1405, USA ACM SIGEVO (formerly ISGEC). **1**: 815--822.
- Hornby, G. S. (2006). ALPS: the age-layered population structure for reducing the problem of premature convergence. Proceedings of the 8th annual conference on Genetic and evolutionary computation. Seattle, Washington, USA, ACM.
- Hornby, G. S. (2009). Steady-state ALPS for real-valued problems. Proceedings of the 11th Annual conference on Genetic and evolutionary computation. Montreal, Quebec, Canada, ACM.
- Hornby, G. S. (2009). A Steady-State Version of the Age-Layered Population Structure EA. Genetic Programming Theory & Practice VII. R. L. Riolo, U.-M. O'Reilly and T. McConaghy. Ann Arbor, Springer: 87-102.
- Ihsan, E., B. Eric, et al. (2005). Interactive estimation of agent-based financial markets models: modularity and learning. Proceedings of the 2005 conference on Genetic and evolutionary computation. Washington DC, USA, ACM.
- Jaeckel, P. M., T. (1998). "A numerical and experimental study of codimension-2 points in a parametrically excited double pendulum." Royal Society of London Proceedings Series A **454**: 3257-3274.
- Jamshidi, N. and B. Palsson (2008). "Formulating genome-scale kinetic models in the post-genome era." Mol Syst Biol **4**.
- Jansen, T. (2002). On the Analysis of Dynamic Restart Strategies for Evolutionary Algorithms. Parallel Problem Solving from Nature — PPSN VII: 33-43.
- Jin, Y. (2005). "A comprehensive survey of fitness approximation in evolutionary computation." Soft Computing Journal **9**(1): 3-12.
- Jin, Y. and J. Branke (2005). "Evolutionary optimization in uncertain environments-a survey." IEEE Transactions on Evolutionary Computation **9**(3): 303-317.
- Jin, Y., M. Olhofer, et al. (2001). Managing approximate models in evolutionary aerodynamic design optimization. Proceedings of the 2001 Congress on

Evolutionary Computation.

- Jin, Y., M. Olhofer, et al. (2002). "A framework for evolutionary optimization with approximate fitness functions." IEEE Transactions on Evolutionary Computation **6**(5): 481-494.
- Jin, Y. and B. Sendhoff (2004). Reducing Fitness Evaluations Using Clustering Techniques and Neural Network Ensembles. Proceedings of the Genetic and Evolutionary Computation Conference.
- Johanson, B. and R. Poli (1998). GP-Music: An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters. Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, Madison, Wisconsin, USA, Morgan Kaufmann San Francisco, CA, USA.
- Kalyanmoy, D. and K. Deb (2001). Multi-Objective Optimization Using Evolutionary Algorithms, John Wiley & Sons, Inc.
- Kauffman, K. J., J. D. Pajeroski, et al. (2002). "Description and analysis of metabolic connectivity and dynamics in the human red blood cell." Biophysical Journal **83**(2): 646-662.
- Keijzer, M. (2003). Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. Genetic Programming, Proceedings of EuroGP'2003, Essex, Springer-Verlag Berlin EvoNet.
- Kell, D. B. (2004). "Metabolomics and systems biology: making sense of the soup." Current Opinion in Microbiology **7**(3): 296-307.
- Kell, D. B. (2006). "Metabolomics, modelling and machine learning in systems biology - towards an understanding of the languages of cells. Delivered on 3 July 2005 at the 30th FEBS Congress and 9th IUBMB conference in Budapest." FEBS Journal **273**(5): 873-894.

- Kenneth Alan De, J. (1975). An analysis of the behavior of a class of genetic adaptive systems, University of Michigan: 266.
- King, R. D., J. Rowland, et al. (2009). "The Automation of Science." Science **324**(5923): 85-89.
- King, R. D., K. E. Whelan, et al. (2004). "Functional genomic hypothesis generation and experimentation by a robot scientist." Nature **427**(6971): 247-252.
- Kleijnen, J. P. C. (2006). White Noise Assumptions Revisited: Regression Models and Statistical Designs for Simulation Practice: 1-21.
- Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. IJCAI.
- Kotanchek, M., G. Smits, et al. (2008). Trustable symbolic regression models: using ensembles, interval arithmetic and pareto fronts to develop robust and trust-aware models. Genetic Programming Theory and Practice V: 201-220.
- Koza, J. R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA, USA, MIT Press.
- Koza, J. R. (2001). Reverse engineering of metabolic pathways from observed data using genetic programming. Pacific Symposium on Biocomputing Proceedings. Singapore, River Edge, NJ : World Scientific: 434-445.
- Kulkarni, V. G. (1995). Modeling and analysis of stochastic systems, Chapman & Hall, Ltd.
- Kung, H. T., F. Luccio, et al. (1975). "On Finding the Maxima of a Set of Vectors." J. ACM **22**(4): 469-476.
- Larrañaga, P. and J. A. Lozano (2002). Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, Kluwer Academic Publishers.
- Lawrence, S. and C. L. Giles (2000). Overfitting and neural networks: conjugate gradient and backpropagation. Neural Networks, 2000. IJCNN 2000,

- Proceedings of the IEEE-INNS-ENNS International Joint Conference on.
- Lehman, J. and K. O. Stanley (2010). "Abandoning Objectives: Evolution through the Search for Novelty Alone." Evol Comput: 24.
- Leonelli, S. (2010). "Machine Science: The Human Side." Science **330**(6002): 317.
- Levine, A. J., W. Hu, et al. (2007). "Reconstructing signal transduction pathways. challenges and opportunities." Ann.N.Y.Acad.Sci. **1115**(1): 32-50.
- Liang, Y. and B. Feeny (2008). "Parametric identification of a chaotic base-excited double pendulum experiment." Nonlinear Dynamics **52**(1): 181-197.
- Linden, G., S. Hanks, et al. (1997). Interactive Assessment of User Preference Models: The Automated Travel Assistant. Sixth International Conference on User Modeling, Springer: 67--78.
- Lipson, H. (2007). "Principles of modularity, regularity, and hierarchy for scalable systems." The Journal of Biological Physics and Chemistry **7**(4): 125-128.
- Lotka, A. J. (1925). Elements of physical biology. Baltimore, Williams & Wilkins Co.
- Louis, S. J. and G. J. E. Rawlins (1992). Syntactic Analysis of Convergence in Genetic Algorithms. Foundations of Genetic Algorithms 2, Morgan Kaufmann: 141--151.
- Luke, S. and R. P. Wiegand (2002). When Coevolutionary Algorithms Exhibit Evolutionary Dynamics. Workshop Proceedings of the 2003 Genetic and Evolutionary Computation Conference.
- Maamar, H. and D. Dubnau (2005). "Bistability in the Bacillus subtilis K-state (competence) system requires a positive feedback loop." Mol Microbiol **56**(3): 615-24.
- Mackin, K. J. and E. Tazaki (2000). Unsupervised training of multiobjective agent communication using genetic programming. Knowledge-Based Intelligent Engineering Systems and Allied Technologies, 2000. Proceedings. Fourth

International Conference on.

- Mahadevan, R., J. S. Edwards, et al. (2002). "Dynamic Flux Balance Analysis of Diauxic Growth in Escherichia coli." Biophysical Journal **83**(3): 1331-1340.
- Mahfoud, S. W. (1995). Niching methods for genetic algorithms, University of Illinois at Urbana-Champaign.
- Mark, K., S. Guido, et al. (2007). Trustable Symbolic Regression Models. Genetic Programming Theory and Practice V. L. R. Rick, S. Terence and W. Bill. Ann Arbor, Springer: 203--222.
- Marquet, P. A. (2002). "The search for general principles in ecology." Nature **418**(6899): 723-723.
- McConaghy, T., P. Palmers, et al. (2009). Automated Extraction of Expert Domain Knowledge from Genetic Programming Synthesis Results. Genetic Programming Theory and Practice VI, Springer US: 1-14.
- McKay, B., M. J. Willis, et al. (1995). Using a Tree Structured Genetic Algorithm to Perform Symbolic Regression. First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA. A. M. S. Zalzal. Sheffield, UK, IEE London, UK. **414**: 487--492.
- McPhee, N., B. Ohs, et al. (2008). Semantic Building Blocks in Genetic Programming. Genetic Programming: 134-145.
- Mendes, P. and D. B. Kell (1996). "On the analysis of the inverse problem of metabolic pathways using artificial neural networks." BioSystems **38**(1): 15-28.
- Mendes, P. and D. B. Kell (1998). "Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation." Bioinformatics **14**(10): 869-883.
- Mitchell, T. M. (2009). "Mining Our Reality." Science **326**(5960): 1644-1645.

- Mohammad-Reza, A.-T. and J. Mohammad (1997). Incorporating A-Priori Expert Knowledge in Genetic Algorithms. Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation, IEEE Computer Society.
- Moles, C. G., P. Mendes, et al. (2003). "Parameter estimation in biochemical pathways: A comparison of global optimization methods." Genome Res. **13**(11): 2467-2474.
- Monroy, R., G. Arroyo-Figueroa, et al. (2004). Symbolic Regression Problems by Genetic Programming with Multi-branches. MICAI 2004: Advances in Artificial Intelligence, Springer Berlin / Heidelberg. **2972**: 717-726.
- Moore, J. and B. White (2006). Exploiting Expert Knowledge in Genetic Programming for Genome-Wide Genetic Analysis. Parallel Problem Solving from Nature - PPSN IX: 969-977.
- Mor M, W. A., Gottlieb O (2007). Nonlinear Model Based Estimation of Rigid-Body Motion via an Indirect Measurement of An Elastic Appendage. Proceedings of The 21st ASME Biennial Conference on Mechanical Vibration and Noise. Las Vegas, Nevada, USA.
- Mutoh, A., T. Nakamura, et al. (2003). Reducing execution time on genetic algorithm in real-world applications using fitness prediction: parameter optimization of SRM control. Proceedings of the 2003 Congress on Evolutionary Computation, Canberra, ACT, Australia, IEEE Press.
- Nee, S., N. Colegrave, et al. (2005). "The Illusion of Invariant Quantities in Life Histories." Science **309**(5738): 1236-1239.
- Nemenman, I., G. S. Escola, et al. (2007). "Reconstruction of metabolic networks from high-throughput metabolite profiling data. in silico analysis of red blood cell metabolism." Ann.N.Y.Acad.Sci. **1115**(1): 102-115.



- Nguyen, X. H., R. I. McKay, et al. (2001). Solving the Symbolic Regression Problem with Tree-Adjunct Grammar Guided Genetic Programming: The Comparative Results. The Australian Journal of Intelligent Information Processing Systems. **7**: 114--121.
- Ni, T. C. and M. A. Savageau (1996). "Model assessment and refinement using strategies from biochemical systems theory: Application to metabolism in human red blood cells." Journal of Theoretical Biology **179**(4): 329-368.
- Nielsen, J. and S. Oliver (2005). "The next wave in metabolome analysis." Trends in Biotechnology **23**(11): 544-546.
- Noether, E. (1918). "Invariante Variationsprobleme. Nachr. Akad." Phys(1): 235--257.
- O'Reilly, U.-M. (1994). The Trouble Aspects of a Building Block Hypothesis for Genetic Programming, Santa Fe Institute.
- Ochoa, A. and M. R. Soto Ortiz (1997). Partial evaluation of genetic algorithms. 1st Artificial Intelligence Symposium, Havana, Cuba, Inst. Cibernetica, Matematica y Fisica.
- Ong, Y. S., P. B. Nair, et al. (2003). "Evolutionary optimization of computationally expensive problems via surrogate modeling." AIAA Journal **41**(4): 687-96.
- Pagie, L. and P. Hogeweg (1997). "Evolutionary Consequences of Coevolving Targets." Evolutionary Computation **5**(4): 401--418.
- Parke, G., S. Ryan, et al. (2007). "Algorithms and analyses for maximal vector computation." The VLDB Journal **16**(1): 5-28.
- Parzen, E. (1962). "On Estimation of a Probability Density Function and Mode." The Annals of Mathematical Statistics **33**(3): 1065-1076.
- Pelikan, M. and K. Sastry (2004). Fitness inheritance in the Bayesian optimization algorithm. Proceedings of the Genetic and Evolutionary Computation Conference, Seattle, WA, USA, Springer-Verlag.

- Poli, R. (1996). Genetic Programming for Image Analysis. Genetic Programming 1996: Proceedings of the First Annual Conference. J. R. Koza, D. E. Goldberg, D. B. Fogel and R. L. Riolo. Stanford University, CA, USA, MIT Press: 363--368.
- Poli, R. and S. Cagnoni (1997). Genetic Programming with User-Driven Selection: Experiments on the Evolution of Algorithms for Image Enhancement. Genetic Programming 1997: Proceedings of the Second Annual Conference, Stanford University, CA, USA, Morgan Kaufmann San Francisco, CA, USA.
- Potter, M. A. and K. A. De Jong (2000). "Cooperative coevolution: an architecture for evolving coadapted subcomponents." Evolutionary Computation **8**(1): 1-29.
- Ra, E. V. I, et al. (1999). "Discovery tools for science apps." Commun. ACM **42**(11): 37-41.
- Regis, R. G. and C. A. Shoemaker (2004). "Local function approximation in evolutionary algorithms for the optimization of costly functions." IEEE Transactions on Evolutionary Computation **8**(5): 490-505.
- Regis, R. G. and C. A. Shoemaker (2005). "Constrained global optimization of expensive black box functions using radial basis functions." Journal of Global Optimization **31**(1): 153-171.
- Richter, O., A. BETZ, et al. (1975). "Response of Oscillating Glycolysis to Perturbations in Nadh-Nad System - Comparison Between Experiments and A Computer Model." BioSystems **7**: 137-146.
- Riolo, R., T. Soule, et al. (2007). Large-Scale, Time-Constrained Symbolic Regression. Genetic Programming Theory and Practice IV, Springer US: 299-314.
- Rissanen, J. (1978). "Modeling by the shortest data description." Automatica **14**: 465-471.

- Ristroph, L., G. J. Berman, et al. (2009). "Automated hull reconstruction motion tracking (HRMT) applied to sideways maneuvers of free-flying insects." J Exp Biol **212**(9): 1324-1335.
- Rosca, J. P. (1995). Towards automatic discovery of building blocks in genetic programming. Working Notes for the AAAI Symposium on Genetic Programming. E. V. Siegel and J. R. Koza. MIT, Cambridge, MA, USA, AAAI 445 Burgess Drive, Menlo Park, CA 94025, USA: 78-85.
- Rosenblatt, M. (1956). "Remarks on some nonparametric estimates of a density function." Ann. Math. Statist. **27**: 832-837.
- Rosin, C. D. (1997). Coevolutionary search among adversaries, University of California at San Diego.
- Rosin, C. D. and R. K. Belew (1997). "New Methods for Competitive Coevolution." Evolutionary Computation **5**(1): 1-29.
- Rousseeuw, P. J. and A. M. Leroy (1987). Robust regression and outlier detection, John Wiley & Sons, Inc.
- Ruoff, P., M. K. Christensen, et al. (2003). "Temperature dependency and temperature compensation in a model of yeast glycolytic oscillations." Biophysical Chemistry **106**(2): 179-192.
- Sano, Y. and H. Kita (2000). Optimization of noisy fitness functions by means of genetic algorithms using history of search. Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, Paris, France, Springer-Verlag.
- Sastry, K., D. E. Goldberg, et al. (2001). Don't Evaluate, Inherit. Proceedings of the Genetic and Evolutionary Computation Conference.
- Schaffer, W. M., S. Ellner, et al. (1986). "Effects of noise on some dynamical models in ecology." Journal of Mathematical Biology **24**(5): 479-523.

- Schmidt, H., M. F. Madsen, et al. (2008). "Complexity reduction of biochemical rate expressions." Bioinformatics **24**(6): 848-854.
- Schmidt, M. and H. Lipson Symbolic Regression of Implicit Equations. Genetic Programming Theory and Practice VII: 73-85.
- Schmidt, M. and H. Lipson (2007). Comparison of tree and graph encodings as function of problem complexity. Proceedings of the Genetic and Evolutionary Computation Conference, London, ACM Press New York, NY, USA.
- Schmidt, M. and H. Lipson (2009). "Distilling Free-Form Natural Laws from Experimental Data." Science **324**(5923): 81-85.
- Schmidt, M. and H. Lipson. (2009). "Distilling Free-Form Natural Laws from Experimental Data - Supporting Online Material." 2010, from <http://www.sciencemag.org/content/324/5923/81/suppl/DC1>.
- Schmidt, M. D. and H. Lipson (2005). Coevolution of Fitness Maximizers and Fitness Predictors  
Proceedings of the Genetic and Evolutionary Computation Conference, Late Breaking Paper.
- Schmidt, M. D. and H. Lipson (2006). Actively probing and modeling users in interactive coevolution. Proceedings of the Genetic and Evolutionary Computation Conference, Seattle, WA, United States, Association for Computing Machinery, New York, NY 10036-5701, United States.
- Schmidt, M. D. and H. Lipson (2006). Co-evolving Fitness Predictors for Accelerating and Reducing Evaluations. Genetic Programming Theory and Practice IV. L. R. Rick, S. Terence and W. Bill. Ann Arbor, Springer. **5**: 113-130.
- Schmidt, M. D. and H. Lipson (2007). Learning noise. GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, London, ACM Press New York, NY, USA.

- Schmidt, M. D. and H. Lipson (2008). "Coevolution of Fitness Predictors." IEEE Transactions on Evolutionary Computation **12**(6): 736-749.
- Shinbrot, T., C. Grebogi, et al. (1992). "Chaos in a double pendulum." American Journal of Physics **60**(6): 491-499.
- Shpitalni, M, et al. (1997). Classification of sketch strokes and corner detection using conic sections and adaptive clustering. New York, NY, ETATS-UNIS, American Society of Mechanical Engineers.
- Sinderen, D., A. Luttinger, et al. (1995). "comK encodes the competence transcription factor, the key regulatory protein for competence development in *Bacillus subtilis*." Molecular Microbiology **15**(3): 455-462.
- Smith, R. E., B. A. Dike, et al. (1995). Fitness inheritance in genetic algorithms. Proceedings of the ACM Symposium on Applied Computing, Nashville, TN, USA, ACM, New York, NY, USA.
- Smits, G. and M. Kotanchek (2004). Pareto-Front Exploitation in Symbolic Regression. Genetic Programming Theory and Practice {III}. U.-M. O'Reilly, T. Yu, R. L. R. and and B. Worzel. Ann Arbor, Springer: 283-299.
- Smits, W. K., C. C. Eschevins, et al. (2005). "Stripping *Bacillus*: ComK auto-stimulation is responsible for the bistable response in competence development." Mol Microbiol **56**(3): 604-14.
- Smolen, P. (1995). "A model for glycolytic oscillations based on skeletal muscle phosphofructokinase kinetics." Journal of Theoretical Biology **174**(2): 137-148.
- Soule, T. and R. B. Heckendorn (2001). Function Sets in Genetic Programming. Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco, California, USA, Morgan Kaufmann San Francisco, CA 94104, USA.

- Stanley, K. O. and R. Miikkulainen (2004). "Competitive coevolution through evolutionary complexification." Journal of Artificial Intelligence Research **21**: 63-100.
- Stolovitzky, G. and A. Califano (2007). Reverse engineering biological networks : opportunities and challenges in computational methods for pathway inference. Boston, Mass., Blackwell Publishing.
- Strogatz, S. H. (1994). Nonlinear dynamics and chaos, Addison-Wesley Reading, MA.
- Strogatz, S. H. (2001). "Exploring complex networks." Nature **410**(6825): 268-276.
- Styczynski, M. P. and G. Stephanopoulos (2005). "Overview of computational methods for the inference of gene regulatory networks." Computers & Chemical Engineering **29**(3): 519-534.
- Süel, G. M., J. Garcia-Ojalvo, et al. (2006). "An excitable gene regulatory circuit induces transient cellular differentiation." Nature **440**(7083): 545-550.
- Suel, G. M., R. P. Kulkarni, et al. (2007). "Tunability and Noise Dependence in Differentiation Dynamics." Science **315**(5819): 1716-1719.
- Szalay, A. and J. Gray (2006). "2020 Computing: Science in an exponential world." Nature **440**(7083): 413-414.
- Takagi, H. (2001). "Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evaluation." Proceedings of the IEEE **89**(9): 1275--1296.
- Tegner, J., M. K. S. Yeung, et al. (2003). "Reverse engineering gene networks: Integrating genetic perturbations with dynamical modeling." Proceedings of the National Academy of Sciences **100**(10): 5944-5949.
- Termonia, Y. and J. Ross (1981). "Oscillations and Control Features in Glycolysis: Numerical Analysis of a Comprehensive Model." Proceedings of the National Academy of Sciences **78**(5): 2952-2956.

- Terrell, G. and D. Scott (1992). "Variable kernel density estimation." The Annals of Statistics **20**(3): 1236-1265.
- Touretzky, D. S., T. K. Leen, et al. (2007). Learning Local Error Bars for Nonlinear Regression.
- Uday, K. C. and Z. J. Cezary (2003). "An analysis of Gray versus binary encoding in genetic search." Inf. Sci. **156**(3-4): 253-269.
- van Someren, E. P., B. L. T. Vaes, et al. (2006). "Least absolute regression network analysis of the murine osteoblast differentiation network." Bioinformatics **22**(4): 477-484.
- Varma, A. and B. O. Palsson (1994). "Metabolic Flux Balancing: Basic Concepts, Scientific and Practical Use." Nat.Biotechnol. **12**(10): 994-998.
- Volterra, V. (1926). "Variazioni e fluttuazioni del numero d'individui in specie animali conviventi." Mem. R. Accad. Naz. dei Lincei **2**(VI).
- Vugrin, K. W., L. P. Swiler, et al. (2007). "Confidence region estimation techniques for nonlinear regression in groundwater flow: Three case studies." Water Resour. Res. **43**(3): W03423.
- Waltz, D. and B. G. Buchanan (2009). "Automating Science." Science **324**(5923): 43-44.
- Watson, R. A. and J. B. Pollack (2001). Coevolutionary Dynamics in a Minimal Substrate. Proceedings of the Genetic and Evolutionary Computation Conference.
- Wolf, J. and R. Heinrich (2000). "Effect of cellular interaction on glycolytic oscillations in yeast: a theoretical investigation." Biochem. J. **345**(2): 321-334.
- X. Wen, S. F. R. S. (1999). Linear Modeling Of mRNA Expression Levels During CNS Development And Injury, unknown.
- Xavier, L. and E. G. David (2003). "Bounding the effect of noise in multiobjective

- learning classifier systems." Evol. Comput. **11**(3): 279-298.
- Xu, Q. and M. Lipson (2006). "Carrier-induced optical bistability in silicon ring resonators." Opt. Lett. **31**(3): 341-343.
- Xu, Q. and M. Lipson (2007). "All-optical logic based on silicon micro-ring resonators." Opt. Express **15**(3): 924-929.
- Xu, Q., S. Sandhu, et al. (2006). Experimental Realization of an On-Chip All-Optical Analogue to Electromagnetically Induced Transparency. Conference on Lasers and Electro-Optics/Quantum Electronics and Laser Science Conference and Photonic Applications Systems Technologies, Optical Society of America.
- Xu, Q., J. Shakya, et al. (2006). "Direct measurement of tunable optical delays on chip analogue to electromagnetically induced transparency." Opt. Express **14**(14): 6463-6468.
- Yang, D. and S. J. Flockton (1995). Evolutionary algorithms with a coarse-to-fine function smoothing. 1995 IEEE International Conference on Evolutionary Computation, Perth, WA, Australia, IEEE.
- Young, J. D. and D. Ramkrishna (2007). "On the Matching and Proportional Laws of Cybernetic Models." Biotechnology Progress **23**(1): 83-99.
- Zhang, Y. and P. Rockett (2007). "A Comparison of three evolutionary strategies for multiobjective genetic programming." Artif. Intell. Rev. **27**(2-3): 149-163.
- Zitzler, E., M. Laumanns, et al. (2001). SPEA2: Improving the Strength Pareto Evolutionary Algorithm.
- Zykov, V., J. Bongard, et al. (2005). Co-evolutionary Variance Can Guide Physical Testing in Evolutionary System Identification. Proceedings of Evolvable Hardware, IEEE Computer Society.