Proceedings of the 9th Biennial ASME Conference on Engineering Systems Design and Analysis
ESDA08
July 7-9, 2008, Haifa, Israel

ESDA2008-59309

DATA-MINING DYNAMICAL SYSTEMS: AUTOMATED SYMBOLIC SYSTEM IDENTIFICATION FOR EXPLORATORY ANALYSIS

Michael D. Schmidt
Computational Biology,
Cornell University
Ithaca, NY 14853, USA

Hod Lipson
School of Mechanical & Aerospace Engineering,
and Department of Computing & Information
Science, Cornell University
Ithaca, NY 14853

ABSTRACT

This paper 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.

1. 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 [1-3]. We represent differential equations as an acyclic graph of primitive operations - such as *abs*, *exp*, and *log*, or binary operations such as *add*, *mult*, and *div*. 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 [3]. 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 [4].

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.

2. SYMBOLIC REGRESSION

Symbolic regression is the problem of identifying the analytical mathematical equation of a hidden system from experimental data [5-8]. Unlike polynomial regression or related machine learning methods that also fit data, symbolic regression is a system identification method, which explicates behavior. Symbolic regression is closely related to general machine learning problems however, it remains an open-ended and discrete problem that cannot be solved greedily.

Symbolic regression uses genetic programming - a widely studied class of algorithms inspired by biological evolution - to

Copyright © 2008 by ASME

evolve (compete) algebraic expressions to explain experimental data [8]. Unlike polynomial regression or neural networks which can also fit data, symbolic regression searches function-space to explain experimental observations by incrementally composing basic algebraic building blocks into increasingly larger equations, 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 system behavior mechanisms.

Genetic programming is an evolutionary algorithm inspired by biological evolution where populations of candidate solutions compete to solve a problem [8]. In a traditional genetic program, an initially random population of solutions evolves iteratively to maximize some fitness objective - for example, 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 evolves populations of algebraic expressions in a similar fashion.

2.1. Fitness Prediction

The fitness objective of a symbolic regression solution is to minimize error on the training set [7, 9-11]. There are many ways to measure the error such as squared error, absolute error, log error, etc. Though the choice is not critical to the optimal solution, different metrics work better on different problems. For experiments in this paper, we use the mean absolute error for fitness measurement:

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

where s is a candidate solution (an algebraic expression), x_i and y_i are training data input and outputs, and N is the total number of training examples in training data set.

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 [1, 3]. In past research, algorithms have used all available data at once to evaluate the fit. However, this metric can be overly stringent – inhibiting solutions from building intermediate expressions needed for the final model. Fitness predictors only measure fit on a small subset of the data. This allows 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.

The fitness predictor is encoded as a small array of indexes to the full training data set. 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 emphasizing particular features). The predicted fitness is calculated as:

$$predicted_fitness(s) = \frac{1}{n}\sum_{i=1}^{n}\left|s(x_i) - y_i\right|$$

where s is a candidate solution (algebraic expression), x_i and y_i are training data inputs and outputs in the training dataset indexed by the predictor, and n is the number of samples in the predictor.

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

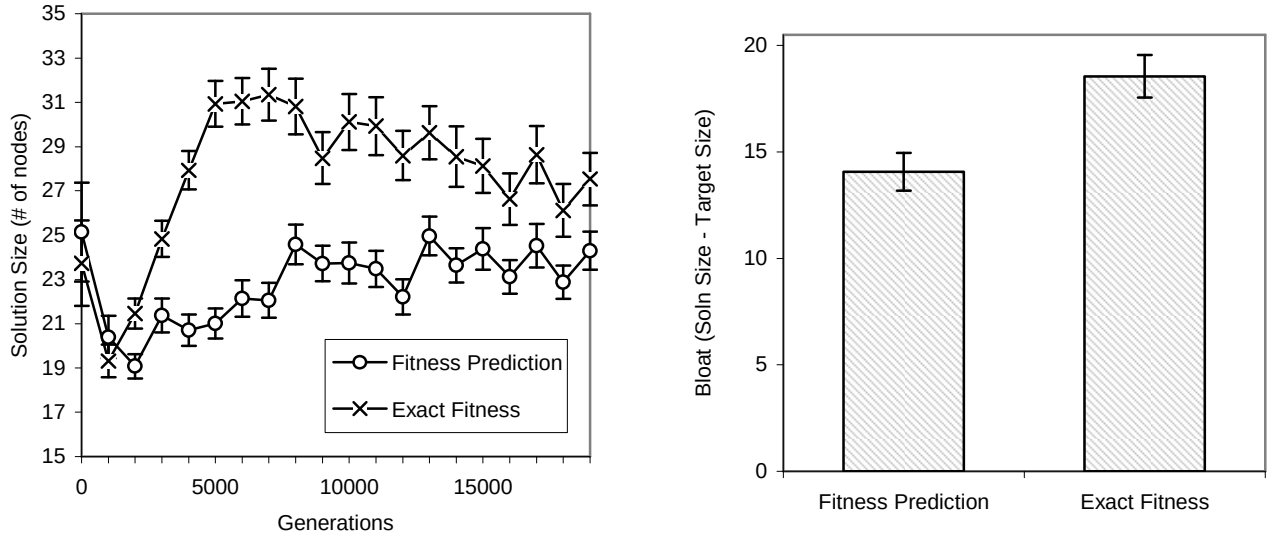


Figure 1. The size (number of nodes in the equation) of the best solution during regression averaged over 100 test runs (left pane). The right pane shows the total average bloat averaged over 500 randomly generated expressions. Error bars show the standard error.

Copyright © 2008 by ASME

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.

2.2. Reducing Function 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 1 shows the size of the best solution during evolution on the benchmark function $y=e^{|x|}sin(2x)$ averaged over 100 test runs. The benchmark function 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.

On average, fitness prediction maintains significantly less complex solutions during evolution than the algorithm using exact fitness calculations. The exact fitness solutions balloon near 5000 generations while fitness prediction experiences solution sizes that are both lower and more consistent.

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. The right pane of Figure 1 shows the bloat of final solutions of both algorithms on 500 randomly generated target functions.

Fitness prediction yields less bloated solutions on average for randomly generated functions. 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.

2.3. Comparison to Previously Published Methods

In this section, we compare the fitness prediction algorithm with four recently published symbolic regression techniques: Stepwise Adaptive Weights (SAW) [10], Grammar Guided Genetic Programming (GGGP) [7], Tree-Adjunct Grammar Guided Genetic Programming (TAG3P) [7], Coevolution with Tractable Shared Fitness [9], Distinction Fitness [9], and random sampling [9].

We compare computational performance based on *point evaluations*, defined by the total number of times the output of any symbolic expression is evaluated. The fitness prediction 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 fitness prediction algorithm to calculate fitness for every generation, even though different selection algorithms do not require it.

Table I. Symbolic regression with fitness prediction compared to other published methods

Algorithm	Target Function§	Metric§	Published Results	Fitness Prediction
PSAW	$f(x) = x^5 - 2x^3 + x$	Convergence†	85.9%	**93.9%**
	$f(x) = x^6 - 2x^4 + x^2$		81.8%	**86.9%**
GGGP	*P2, P3, P4, P5**	Convergence††	92%, 64%, 48%, 28%	**100%, 86%, 62%, 52%**
	$f(x) = 1 - 2\sin^2(x)$**		20%	**76%**
TAG3P	*P2, P3, P4, P5**	Convergence††	100%, **100%, 96%, 84%**	100%, 86%, 62%, 52%
	$f(x) = 1 - 2\sin^2(x)$**		36%	**76%**
Coevolved Tractable	Gaussian	Evaluations†††	3.384e7	**2.107e7**
Coevolved Distinction	Gaussian		5.070e7	
Random Sampling	Gaussian		6.006e8	

* 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

Copyright © 2008 by ASME

Many of these experiments were carried out 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 I are shown in bold text. The fitness prediction 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. Fitness prediction makes a 40% improvement in convergence for the trigonometric identity experiment. The comparison with coevolved tractable, shared, and random sampling algorithms show fitness prediction 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 are past the fringe [9].

3. 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 [12] 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.

Table II. 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 (eg. *x*, *y*)
Crossover	variable position, single point
Fitness Predictor Sample Size	16

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 [13-18]. However if prior knowledge is limited, it may not be possible to model the system mathematically beyond simple linear models with standard methods [19]. 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.

3.1. 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.

3.2. 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 [20] 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 [4] by incorporating random variables into the mathematical model.

3.3. Finding Time 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 [20] 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:

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

Copyright © 2008 by ASME

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 inetgrating 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.

4. RESULTS & 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 [21]. These systems exhibit many remarkable dynamics (eg. bi-stability, hysteresis, limit cycles) and are frequently used to understand behavior of other related systems.

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 the 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 [22]. Every 100 generations, we reshuffle all solutions across all populations. Table II 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

Table III. Inferring various physical and biological dynamical models

	System	Inferred	Time	Point Evals
Glider	$\dot{v} = -0.05 \cdot v^2 - \sin(\theta)$	$\dot{v} = -0.0499999 \cdot v^2 - \sin(\theta)$	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
Bacterial Respiration	$\dot{x} = 20 - x - \frac{x \cdot y}{1 + 0.5 \cdot x^2}$	$\dot{x} = 19.994 - 0.998 \cdot x - \frac{1.999 \cdot y}{x + 1.995/x}$	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
Predator-Prey	$\dot{x} = x \cdot \left(4 - x - \frac{y}{1 + x}\right)$	$\dot{x} = 4.002 \cdot x - x^2 - \frac{1.003 \cdot x \cdot y}{1.003 + x}$	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
Bar Magnets	$\dot{\theta}_1 = 0.5 \cdot \sin(\theta_1 - \theta_2) - \sin(\theta_1)$	$\dot{\theta}_1 = -\sin(\theta_1) - 0.5 \cdot \sin(\theta_2 - \theta_1)$	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
Shear Flow	$\dot{\theta} = \cot\phi \cos\theta$	$\dot{\theta} = \frac{\cos\phi}{\sin\phi} \cdot \cos\theta$	3.562 sec	0.36 B
	$\dot{\phi} = (\cos^2\phi + 0.1 \cdot \sin^2\phi) \cdot \sin\theta$	$\dot{\phi} = 0.099 \cdot \sin\theta + 0.9 \cdot \sin\theta \cos\phi \cos\phi$	33.859 sec	3.42 B
van der Pol	$\dot{x} = 10 \cdot \left[y - \left(\frac{1}{3} \cdot x^3 - x\right)\right]$	$\dot{x} = 9.999 \cdot x + 9.999 \cdot y - 3.333 \cdot x^3$	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
Lotka-Volterra	$\dot{x} = 3 \cdot x - 2 \cdot x \cdot y - x^2$	$\dot{x} = 3 \cdot x - x^2 - 2 \cdot x \cdot y$	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

Copyright © 2008 by ASME

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 III.

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.

5. 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 has allowed us to infer a number of nonlinear physical and biological systems directly from data.

6. ACKNOWLEDGMENTS

This research is supported in part by the Cornell IGERT Program in Nonlinear Systems and the US National Science Foundation.

7. REFERENCES

[1] M. D. Schmidt and H. Lipson, "Co-evolving Fitness Predictors for Accelerating and Reducing Evaluations," in *Genetic Programming Theory and Practice IV*. vol. 5, L. R. Rick, S. Terence, and W. Bill, Eds. Ann Arbor: Springer, 2006, pp. -.

[2] M. D. Schmidt and H. Lipson, "Actively probing and modeling users in interactive coevolution," in *Proceedings of the Genetic and Evolutionary Computation Conference*, Seattle, WA, United States, 2006, pp. 385-386.

[3] M. D. Schmidt and H. Lipson, "Coevolution of Fitness Predictors," *IEEE Transactions on Evolutionary Computation*, vol. in press, 2008.

[4] D. S. Michael and L. Hod, "Learning noise," in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, London, 2007, pp. 1680--1685.

[5] D. A. Augusto and H. J. C. Barbosa, "Symbolic Regression via Genetic Programming," in *VI Brazilian Symposium on Neural Networks (SBRN'00)*, Rio de Janeiro, RJ, Brazil, 2000, p. 173.

[6] J. Duffy and J. Engle-Warnick, "Using Symbolic Regression to Infer Strategies from Experimental Data," *Evolutionary Computation in Economics and Finance*, vol. 100, pp. 61--84, 2002.

[7] N. X. Hoai, R. I. McKay, D. Essam, and R. Chau, "Solving the Symbolic Regression Problem with Tree-Adjunct Grammar Guided Genetic Programming: The Comparative Results," in *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002, pp. 1326--1331.

[8] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[9] B. Dolin, F. H. B. III, and E. G. Rieffel, "Co-evolving an effective fitness sample: experiments in symbolic regression and distributed robot control," in *Proceedings of the 2002 ACM symposium on Applied computing*, Madrid, Spain, 2002, pp. 553-559.

[10] J. Eggermont and J. I. v. Hemert, "Stepwise Adaptation of Weights for Symbolic Regression with Genetic Programming," in *Proceedings of the Twelveth Belgium/Netherlands Conference on Artificial Intelligence (BNAIC'00)*, De Efteling, Kaatsheuvel, Holland, 2000, pp. 259--266.

[11] M. Keijzer, "Improving Symbolic Regression with Interval Arithmetic and Linear Scaling," in *Genetic Programming, Proceedings of EuroGP'2003*, Essex, 2003, pp. 70--82.

[12] M. Bansal, V. Belcastro, A. Ambesi-Impiombato, and D. di Bernardo, "How to infer gene networks from expression profiles," *Mol Syst Biol*, vol. 3, 2007.

[13] M. Bansal, G. D. Gatta, and D. di Bernardo, "Inference of gene regulatory networks and compound mode of action from time course gene expression profiles," *Bioinformatics*, vol. 22, pp. 815-822, April 1, 2006 2006.

[14] R. Bonneau, D. Reiss, P. Shannon, M. Facciotti, L. Hood, N. Baliga, and V. Thorsson, "The Inferelator: an algorithm for learning parsimonious regulatory

 Copyright © 2008 by ASME

networks from systems-biology data sets de novo," *Genome Biology,* vol. 7, p. R36, 2006.

[15] D. di Bernardo, M. J. Thompson, T. S. Gardner, S. E. Chobot, E. L. Eastwood, A. P. Wojtovich, S. J. Elliott, S. E. Schaus, and J. J. Collins, "Chemogenomic profiling on a genome-wide scale using reverse-engineered gene networks," *Nat Biotech,* vol. 23, pp. 377-383, 2005.

[16] T. S. Gardner, D. di Bernardo, D. Lorenz, and J. J. Collins, "Inferring Genetic Networks and Identifying Compound Mode of Action via Expression Profiling," *Science,* vol. 301, pp. 102-105, July 4, 2003 2003.

[17] J. Tegner, M. K. S. Yeung, J. Hasty, and J. J. Collins, "Reverse engineering gene networks: Integrating genetic perturbations with dynamical modeling," *Proceedings of the National Academy of Sciences,* vol. 100, pp. 5944-5949, May 13, 2003 2003.

[18] E. P. van Someren, B. L. T. Vaes, W. T. Steegenga, A. M. Sijbers, K. J. Dechering, and M. J. T. Reinders, "Least absolute regression network analysis of the murine osteoblast differentiation network," *Bioinformatics,* vol. 22, pp. 477-484, February 15, 2006 2006.

[19] S. F. R. S. X. Wen, "Linear Modeling Of mRNA Expression Levels During CNS Development And Injury," unknown, 1999.

[20] W. S. Cleveland and S. J. Devlin, "Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting," *Journal of the American Statistical Association,* vol. 83, pp. 596-610, 1988.

[21] S. H. Strogatz, *Nonlinear dynamics and chaos*: Addison-Wesley Reading, MA, 1994.

[22] F. Francisco, T. Marco, and V. Leonardo, "An Empirical Study of Multipopulation Genetic Programming," in *Genetic Programming and Evolvable Machines.* vol. 4, 2003, pp. 21--51.

Copyright © 2008 by ASME