Coevolution of Fitness Predictors

Michael D. Schmidt and Hod Lipson, *Member, IEEE*

***Abstract*—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.**

***Index Terms*—Bloat Reduction, coevolution, fitness modeling, symbolic regression.**

I. Introduction

FITNESS PREDICTION is a technique used to replace fitness evaluations in evolutionary algorithms with a lightweight approximation that adapts with the solution population. A closely related concept to fitness prediction is *fitness modeling*, where a predefined model or coarse simulation is used to approximate fitness in cases where the exact fitness requires an expensive simulation or physical experiment [1], [2]. 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 fitnesses to continuous values, and diversifying populations through ambiguity [3]. In this paper, 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 [3]. 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 paper 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 fitnesses.

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

II. Related Work

A. 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 rel-

Manuscript received September 5, 2006; revised January 29, 2007July 27, 2007November 11, 2007. First published March 20, 2008; current version published December 2, 2008. This research was supported in part by the U.S. National Science Foundation (NSF) under Grant DMI 0547376.

M. D. Schmidt is with the Department of Computer Science, Cornell University, Ithaca, NY 18450 USA (e-mail: mds47@cornell.edu).

H. Lipson is with the Department of Mechanical and Aerospace Engineering and the Department of Computing and Information Science, Cornell University, Ithaca NY 14853–7501 USA (e-mail: hod.lipson@cornell.edu).

Digital Object Identifier 10.1109/TEVC.2008.919006

ative fitness ranking between two other individuals in the same or a separate population [4]. 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.

Several studies have been devoted to the application of coevolution to enhance problem solving [5]–[13], with the main goal of controlling coevolutionary dynamics that often result in a lack of progress or progress in unanticipated directions [14]–[18]. Here, we use a specific form of coevolution [19], [20] 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 (for other benefits, see Section VI). The fitness predictor population benefits from the solution population by refining its approximation in the most useful areas of the fitness domain.

B. Fitness Modeling

Fitness modeling has become an active area in evolutionary computation with many varying approaches and results [3]. Here, we discuss the motivations, methods, and challenges of fitness modeling.

1) 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 multimodal landscapes, and define a continuous fitness in domains that lack an explicit fitness (e.g., evolving art and music) [3]. Here, we discuss motivations for fitness modeling and example applications.

- **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 [1], [2], [21]–[25] that often requires time-consuming finite-element calculations. Often the resolution provided by the exact fitness objective is unnecessary for evolutionary progress.
- **No explicit fitness**: Many domains do not have a computable fitness. For example, in human interactive evolution [26] (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 [11], [27], [28].
- **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 [29]. An alternative approach may be to develop a statistical model [30].
- **Smoothing landscapes**: Almost all evolutionary domains suffer from multimodal 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 [24], [25], [31], [32].
- **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 paper.

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.

2) 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 fitnesses efficiently [11], [22]. 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 [3].

Subsampling of training data is also a common way to reduce the cost of fitness evaluation [17], [33]. In many problems, fitness is calculated by evaluating individuals on training cases and combining the total error. With a subsample, 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 [34]–[36], 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 [37] (EDAs) builds a model of the fitness function based on the structure of the probabilistic model used in the algorithm [38]. In fitness imitation [22], 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 [39], 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 [3]. For example, a fitness predictor such as a neural network is used to select offspring from all potential crossovers of two parents [23]. In this

paper, however, we focus only on replacing actual fitness evaluations with the fitness predictor.

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

- **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 fitnesses 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 [32], 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.
- **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.
- **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 [19]. 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.

III. Coevolved Fitness Predictors

A. 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.

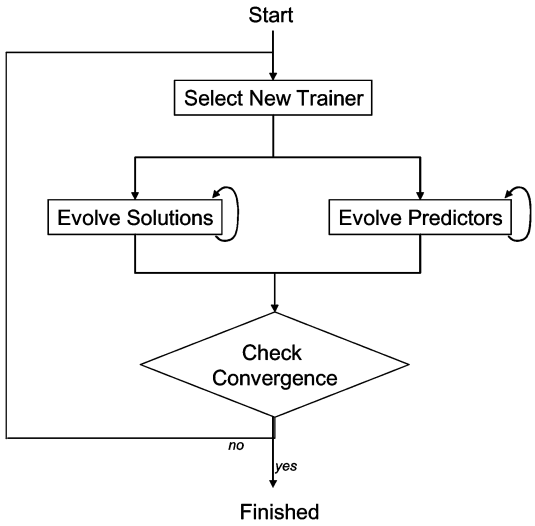

Fig. 1. High-level overview of the coevolution of solutions and fitness predictors algorithm.

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

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

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

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

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 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 fitnesses of the current population.

Algorithm

1) Overview: The algorithm presented in this paper 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 Fig. 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 shown in Fig. 2.

2) Evaluating Exact Fitnesses: 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 [40] and with improving evolved individuals [19].

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.

3) Evolving the Populations: Candidate solutions and fitness predictors are coevolved in parallel using two threads. Pseudocode is shown in Fig. 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). The algorithm then returns the solution 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.

```
Begin 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 fitnesses for solutions using pred
      Perform selection
      Sort population
      If  top-ranked solution error < epsilon
         Return solution and Exit
      End if
   End repeat
End Thread
```

```
Begin 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 fitnesses 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
End Thread
```

Fig. 2. Pseudocode for the two threads in the algorithm to 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[1]) 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

[1]Here, and elsewhere in this paper, 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.

observed that the ratio is robust over a relatively wide range of values (see Section IV-D).

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.

4) 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 ($< \varepsilon$) error on all training data examples. If the best solution has not converged at this step, a new trainer is added (Fig. 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.

IV. 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.

Symbolic regression is the problem of identifying the analytical mathematical description of a hidden system from experimental data [41], [42]. 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.

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. First, however, we introduce basic concepts of symbolic regression and cover related research.

A. Symbolic Regression Overview

1) Symbolic Regression Encoding: For experiments in this paper, we represent functional expressions as a binary tree of primitive operations [41], [43], [44]. The operations can be unary operations such as *abs*, *exp*, and *log*, or binary operations such as *add*, *sub, mult*, and *div*. If some prior knowledge of the problem is known, the types of operations available can be chosen ahead of time [41], [42], [45]. The terminal values available consist of the function's input variables and the function's evolved constant values [46].

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 subtrees 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.

The fitness objective of a symbolic regression solution is to minimize error on the training set [43], [47]–[49]. 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

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

where s is a candidate solution (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.

2) Coevolution in Symbolic Regression: Coevolving training examples is a well-studied approach in symbolic regression [17], [47]. 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.

B. Subsample Fitness Predictors

1) 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

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

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.

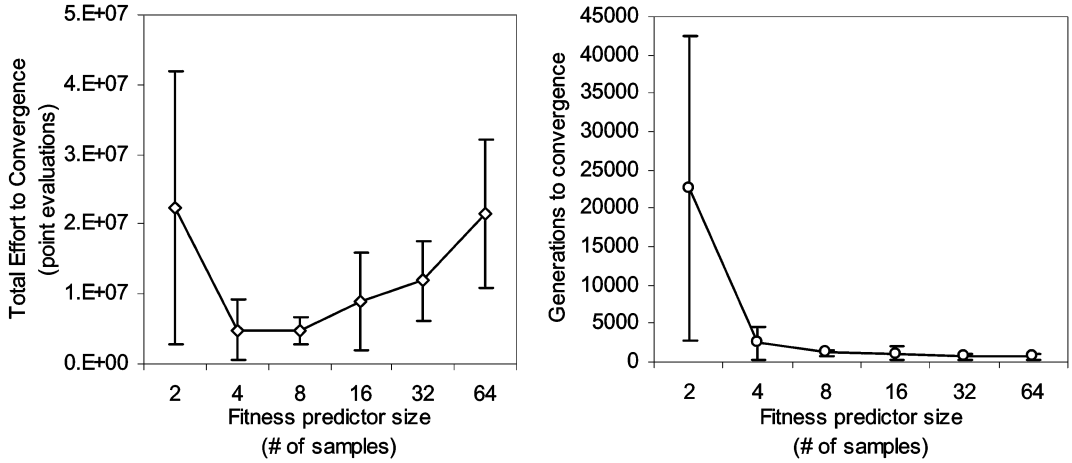

Fig. 3. The expected point evaluations before convergence versus the number of samples in the fitness predictor. Error bars show the standard deviation.

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.

2) 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 a 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 nonlinear function that has two local minima approximations that make finding the exact solution difficult. In the following sections, we also use this function as a benchmark for some empirical experimentation because, although it evolves rapidly, it is clearly nontrivial.

When the fitness predictor only has two samples, fitness evaluations are extremely lightweight but the evolutionary process requires many more generations, as evident in Fig. 3. The larger subsets are sufficiently large for accurate modeling but do not greatly reduce the number of generations needed. Fig. 3 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 paper.

In our symbolic regression experiments, we use an eight-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 multivariable functions involved in ongoing research.

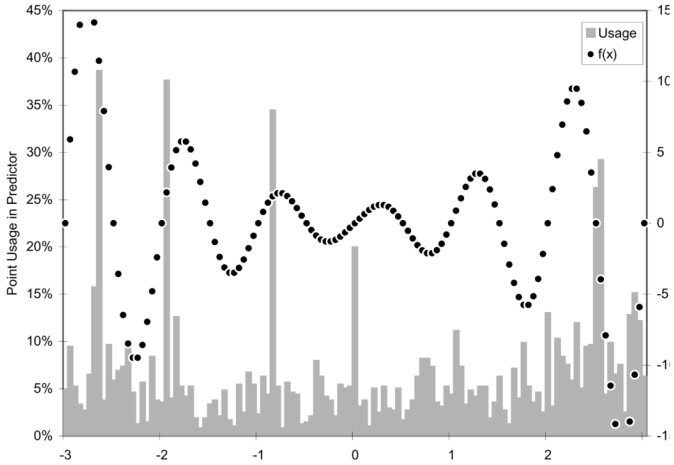

Fig. 4. Histogram of training samples selected by the best fitness predictor during evolution to convergence of $f = e^{|x|}\sin(x)$. Some samples are selected significantly more often that others.

3) 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.

Fig. 4 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 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.

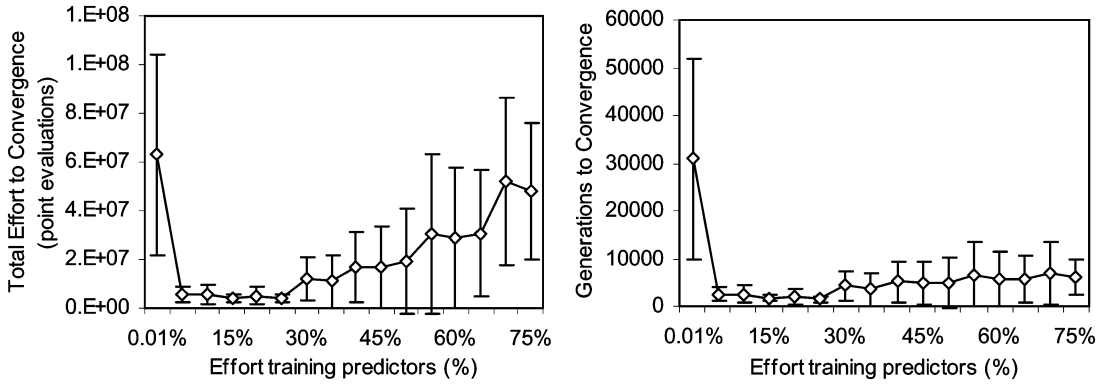

Fig. 5. The expected number of point evaluations before convergence versus the effort (percent of point evaluations), while training the fitness predictors averaged over 50 trials. Error bars show the standard error.

TABLE I
SUMMARY OF THE COMPARED FITNESS PREDICTION STRATEGIES

Strategy	Sample Size	Sample Selection Method
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

C. Experiment 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 [50], 0.1 mutation probability, and 0.5 crossover probability.

The operator set is $\{+, -, *, /, \exp, \log, \sin, \cos\}$ 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).

D. 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.

Fig. 5 shows the impact that the effort ratio has on convergence time with the test function $f = e^{|x|}\sin(x)$. Ratios in the range 5%–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. We discuss specific values for a sample application in Section V.

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 lightweight, they should require only a fraction of the effort that the solutions require.

V. EXPERIMENTAL RESULTS

A. Examining Behavior on Test Problems

Here, we compare four fitness algorithms in symbolic regression listed in Table I. 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 rerandomized 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.

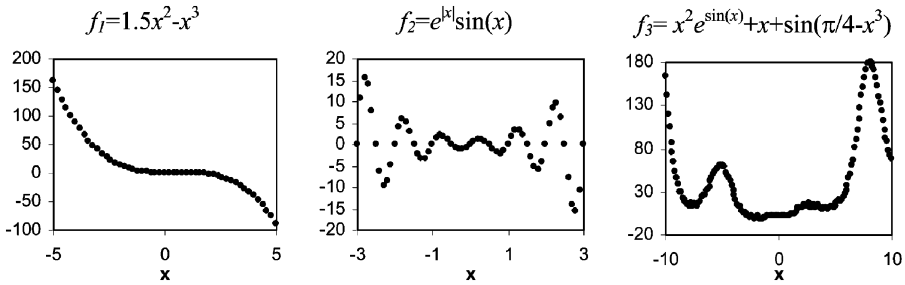

Fig. 6. The training data of the three target functions experimented on. The horizontal axis shows the input values x. The vertical axis shows the output training value $f(x)$.

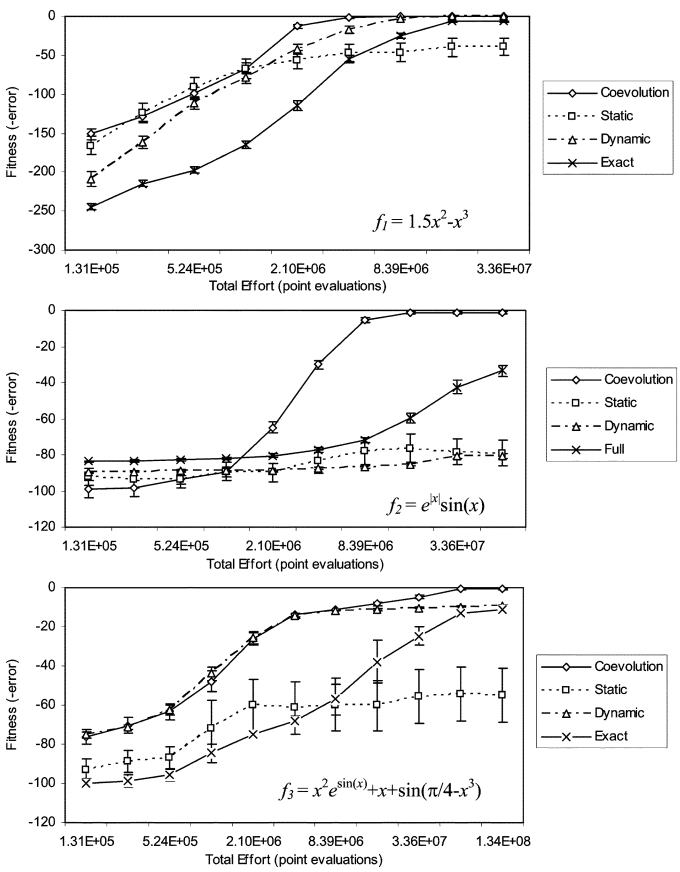

Fig. 7. The test set fitness during evolution for target functions $f_1(x)$, $f_2(x)$, and $f_3(x)$, respectively. Results are averaged over 50 trials. Error bars show the standard error.

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 [43], [47]–[49].

In this section, we test on three different target functions that elicit different behaviors from the four algorithms. The training data, shown in Fig. 6, 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 Fig. 7.

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.

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 subsample or a randomly changing subsample. Thus, the effect of coevolution must play an important role.

Later in this paper, we compare the convergence rates of these algorithms over randomly generated functions to observe more general trends.

B. Comparison to Previously Published Methods

In this section, we compare the coevolution algorithm with four recently published symbolic regression techniques: Stepwise Adaptive Weights (SAWs) [43], Grammar Guided Genetic Programming (GGGP) [48], Tree-Adjunct Grammar Guided Genetic Programming (TAG3P) [48], Coevolution With Tractable Shared Fitness [47], Distinction Fitness [47], and Random Sampling [47]. We did not reimplement 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.

TABLE II
PERFORMANCE COMPARISON TO PUBLISHED METHODS

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

* 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

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 II 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 are past the fringe [12].

Next, we make an empirical comparison with fitness inheritance [34]–[36]. As mentioned in Section II-B2, fitness inheritance is a fitness prediction approach that evaluates exact fitnesses 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).

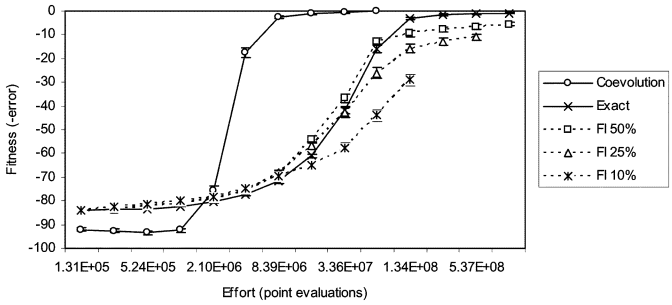

Fig. 8. Test set fitness versus evaluations averaged over 100 test runs for $f_2(x)$. Error bars show standard error.

Fig. 8 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 [21], 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 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 fitnesses can yield performance increases with similar convergence rates.

C. Testing Scalability Using Randomly Generated Symbolic Functions

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.

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 III.

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

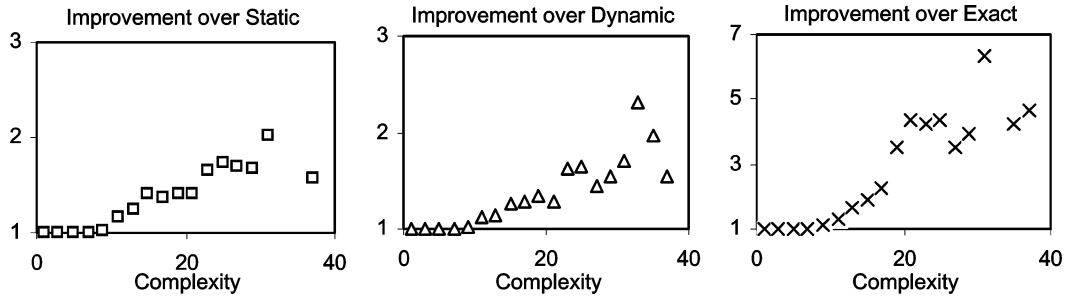

Fig. 10. Improvement factor in convergence of coevolution over the other algorithms versus complexity for random target functions.

TABLE III
EXAMPLES OF RANDOMLY GENERATED FUNCTIONS AND THEIR COMPLEXITIES

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

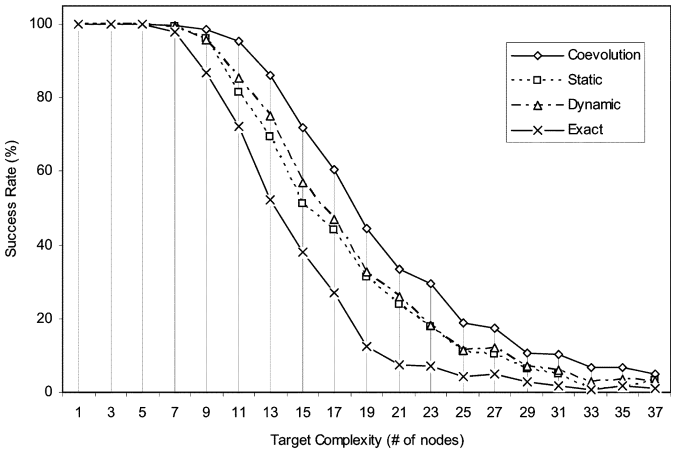

Fig. 9. The percent of successful convergence after 10 M point evaluations versus the target function complexity (the number of nodes in the binary expression tree).

complexities from 1–40. This yields approximately 150 random trials of complexity.

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 Fig. 9.

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 (see Appendix B).

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 fitnesses, 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 pairwise 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 (Fig. 9), the improvement factor for coevolution tends to increase (Fig. 10). Statistical testing (see Appendix B) 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.

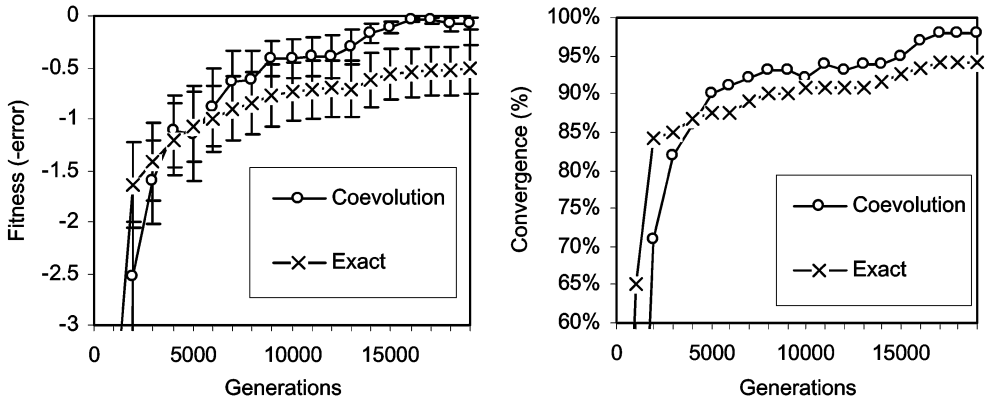

Fig. 11. Fitness and percent of runs converged versus generations throughout evolution on the function $f_2(x)$. Error bars show the standard error. Note that exact evaluations are performing significantly more computational effort per generation.

VI. 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.

A. 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.

Fig. 11 shows the performance of each algorithm over 20 000 generations, while regressing $f_2(x)$. This is sufficiently long 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 fitnesses. There are several possible explanations for this. Fitness approximation can drive solutions to unexplored areas of the domain [25], [51], perhaps increasing convergence. Additionally, adapting the fitness approximation can destabilize local optima solutions, as also noted earlier [3], [17]. 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 machine learning 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 paper.

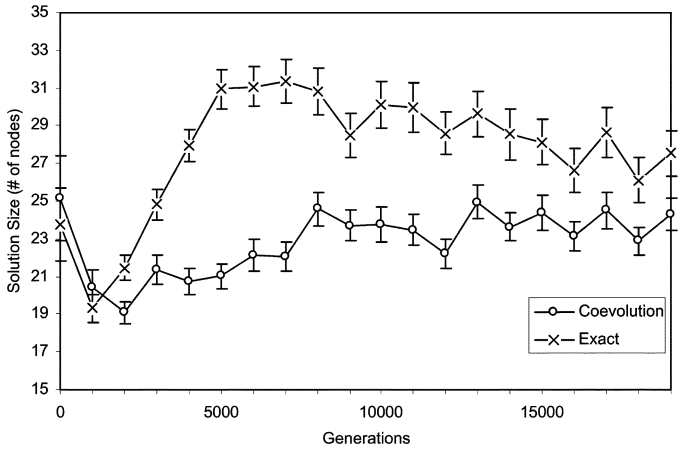

Fig. 12. The size of the best solution during evolution of $f_2(x)$ averaged over 100 test runs. Error bars show the standard error.

B. 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. Fig. 12 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 cause 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

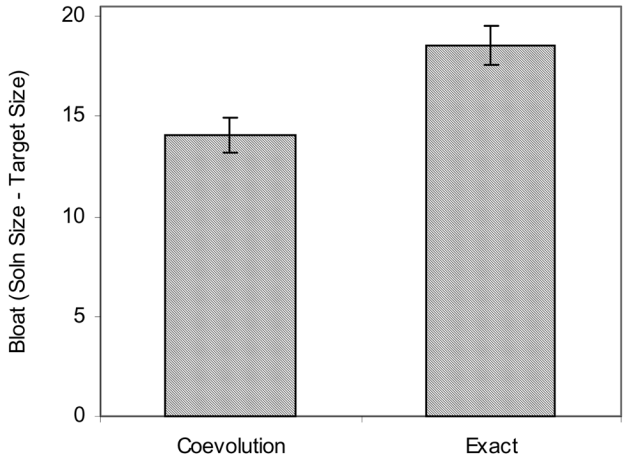

Fig. 13. The bloat of final converged solutions averaged over 500 randomly generated target functions. Error bars show the standard error.

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

Fig. 14. Function simplification pseudocode.

calculations. The exact fitness solutions balloon near 5000 generations, while coevolution experiences solution sizes that are both lower and more consistent.

This preliminary result from $f_2(x)$ suggests 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.

Fig. 13 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

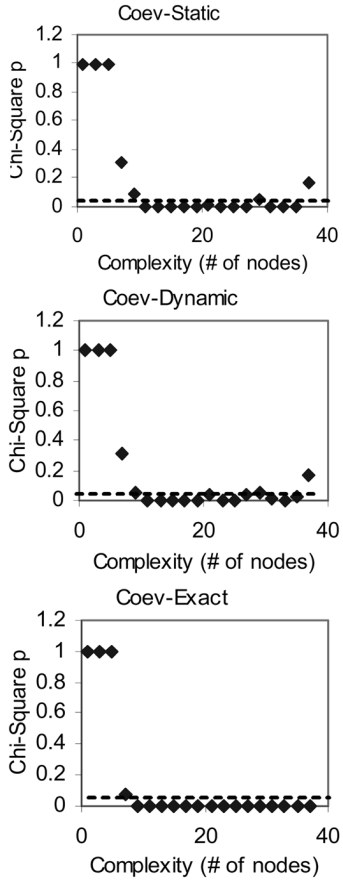

Fig. 15. Chi-square significance of convergence verses complexity.

TABLE IV
CHI-SQUARE SIGNIFICANCE OF CONVERGENCE RATES

	Chi-Square p		
Complexity	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	**1.54E-05**
11	**1.08E-05**	**0.000536**	**1.96E-10**
13	**9.56E-06**	**0.002441**	**7.75E-17**
15	**4.1E-07**	**0.000281**	**9.57E-18**
17	**3.92E-05**	**0.001073**	**5.17E-20**
19	**0.000431**	**0.001726**	**4.75E-32**
21	**0.007439**	**0.040599**	**2.57E-34**
23	**0.000303**	**0.000303**	**4.84E-25**
25	**0.001503**	**0.004607**	**4.32E-16**
27	**0.002755**	**0.044423**	**1.91E-13**
29	**0.049535**	**0.049535**	**1.19E-09**
31	**0.003649**	**0.0161**	**2.14E-23**
33	**1.71E-19**	**0.002359**	**1.71E-19**
35	**1.23E-08**	**0.022948**	**1.23E-08**
37	0.172386	0.172386	**1.94E-05**

also improve computational performance *per point evaluation*, since smaller expressions can be evaluated faster.

Coevolutionary bloat reduction is an important observation for this paper, 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.

VII. Conclusion

In this paper, we have proposed coevolution to address three fundamental challenges faced when using fitness prediction 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.

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 fitnesses 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.

Scaling. In experimentation on randomly generated benchmarks, coevolution shows higher performance over all solution complexities tested. The factor of improvement increases as complexity rises.

Performance by generation. Empirical results show that coevolving fitness predictors can yield higher fitness solutions compared with the exact fitness algorithm even when disregarding savings in computational effort. This suggests that the transformation of the fitness landscape is in itself beneficial.

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.

In future work, we are interested in the further exploration of the effects on bloat reduction and reliability in convergence improvements. More specifically, we want to examine the relationship between convergence to local optima and how quickly the fitness approximation varies.

Appendix

Random Symbolic Function Generator: We generated random target functions by building a random tree of operations. The 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 ($[-2, 2]$ in our case), 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 III.

Convergence and Complexity Statistical Significance: Section V-C shows the convergence rate (%) versus the complexity (the number of nodes in the binary expression tree) for randomly generated target functions. The coevolution algorithm shows higher convergence for complexities over nine.

Here, we measure the statistical significance between the convergence rates using the Chi-Square Test. The Chi-Square Test compares the number of converged runs and nonconverged runs of two algorithms in a contingency table. The returned p-value is the probability that the difference could be due to random chance. The chi-square p-values comparing coevolution with each other algorithm are listed in Table IV

A chi-square $\mathrm{p-value} < 0.05$ 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.

References

[1] Y. S. Ong, P. B. Nair, and A. J. Keane, "Evolutionary optimization of computationally expensive problems via surrogate modeling," *AIAA Journal*, vol. 41, pp. 687–96, 2003.

[2] Y. Jin, M. Olhofer, and B. Sendhoff, "Managing approximate models in evolutionary aerodynamic design optimization," in *Proc. 2001 Cong. Evol. Comput.*, 2001, pp. 592–599.

[3] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing Journal*, vol. 9, pp. 3–12, 2005.

[4] W. D. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure," *Artif. Life II*, vol. X, pp. 313–324, 1992.

[5] S. G. Ficici, "Solution concepts in coevolutionary algorithms," in *Comput. Sci.*. Waltham, MA: Brandeis Univ., 2004.

[6] S. G. Ficici and J. B. Pollack, "Pareto optimality in coevolutionary learning," in *Proc. 6th Eur. Conf. Advances in Artif. Life*, 2001, pp. 316–325.

[7] E. D. D. Jong and J. B. Pollack, "Ideal evaluation from coevolution," *Evol. Comput.*, vol. 12, pp. 159–192, 2004.

[8] M. A. Potter and K. A. D. Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evol. Comput.*, vol. 8, pp. 1–29, 2000.

[9] C. D. Rosin, *Coevolutionary Search Among Adversaries*. San Diego, CA: Univ. California, 1997.

[10] C. D. Rosin and R. K. Belew, "New methods for competitive coevolution," *Evol. Comput.*, vol. 5, pp. 1–29, 1997.

[11] M. D. Schmidt and H. Lipson, "Actively probing and modeling users in interactive coevolution," in *Proc. Genetic and Evol. Comput. Conf.*, Seattle, WA, 2006, pp. 385–386.

[12] K. O. Stanley and R. Miikkulainen, "Competitive coevolution through evolutionary complexification," *J. Artif. Intell. Res.*, vol. 21, pp. 63–100, 2004.

[13] V. Zykov, J. Bongard, and H. Lipson, "Co-evolutionary variance can guide physical testing in evolutionary system identification," in *Proc. Evolvable Hardware*, 2005, pp. 213–213.

[14] A. Bucci and J. B. Pollack, "On identifying global optima in cooperative coevolution," in *Proc. Genetic and Evol. Comput. Conf.*, Washington, DC, 2005, pp. 539–544.

[15] D. Cliff and G. F. Miller, "Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations," in *Proc. 3rd Eur. Conf. Advances Artif. Life*, 1995, pp. 200–218.

[16] S. Luke and R. P. Wiegand, "When coevolutionary algorithms exhibit evolutionary dynamics," in *Proc. 2003 Workshop Genetic Evol. Comput. Conf.*, 2002, pp. 236–241.
[17] L. Pagie and P. Hogeweg, "Evolutionary consequences of coevolving targets," *Evol. Comput.*, vol. 5, pp. 401–418, 1997.
[18] R. A. Watson and J. B. Pollack, "Coevolutionary dynamics in a minimal substrate," in *Proc. Genetic Evol. Comput. Conf.*, 2001, pp. 702–709.
[19] J. C. Bongard and H. Lipson, "Nonlinear system identification using coevolution of models and tests," *IEEE Trans. Evol. Comput.*, vol. 9, pp. 361–384, 2005.
[20] J. C. Bongard and H. Lipson, "'Managed challenge' alleviates disengagement in co-evolutionary system identification," in *Proc. Genetic Evol. Comput. Conf.*, Washington, DC, 2005, pp. 531–538.
[21] Y. Jin, M. Olhofer, and B. Sendhoff, "A framework for evolutionary optimization with approximate fitness functions," *IEEE Trans. Evol. Comput.*, vol. 6, pp. 481–494, 2002.
[22] Y. Jin and B. Sendhoff, "Reducing fitness evaluations using clustering techniques and neural network ensembles," in *Proc. Genetic Evol. Comput. Conf.*, 2004, pp. 688–699.
[23] A. Mutoh, T. Nakamura, S. Kato, and H. Itoh, "Reducing execution time on genetic algorithm in real-world applications using fitness prediction: Parameter optimization of SRM control," in *Proc. 2003 Congr. Evol. Comput.*, Canberra, ACT, Australia, 2003, pp. 552–9.
[24] R. G. Regis and C. A. Shoemaker, "Local function approximation in evolutionary algorithms for the optimization of costly functions," *IEEE Trans. Evol. Comput.*, vol. 8, pp. 490–505, 2004.
[25] R. G. Regis and C. A. Shoemaker, "Constrained global optimization of expensive black box functions using radial basis functions," *J. Global Optimization*, vol. 31, pp. 153–171, 2005.
[26] H. Takagi, "Interactive Evol. Comput.: Fusion of the capabilities of EC optimization and human evaluation," *Proc. IEEE*, vol. 89, pp. 1275–1296, 2001.
[27] B. Johanson and R. Poli, "GP-music: An interactive genetic programming system for music generation with automated fitness raters," in *Proc. 3rd Ann. Conf. Genetic Programming*, Madison, WI, 1998, pp. 181–186.
[28] R. Poli and S. Cagnoni, "Genetic programming with user-driven selection: Experiments on the evolution of algorithms for image enhancement," in *Proc. 2nd Ann. Conf. Genetic Programming*, 1997, pp. 269–277.
[29] D. V. Arnold, "Evolution strategies in noisy environments- a survey of existing work," in *Theoretical Aspects of Evolutionary Computing*. Berlin, Germany: Springer-Verlag, 2001, pp. 239–250.
[30] Y. Sano and H. Kita, "Optimization of noisy fitness functions by means of genetic algorithms using history of search," in *Proc. 6th Int. Conf. Parallel Problem Solving from Nature*, Paris, France, 2000, pp. 571–80.
[31] C. Audet, J. J. E. Dennis, D. W. Moore, A. Booker, and P. D. Frank, "Surrogate-Model-Based method for constrained optimization," in *Proc. AIAA/USAF/NASA/ISSMO Symp. Multidisciplinary Anal. Opt.*, 2000, Paper No. AIAA-2000-4891.
[32] D. Yang and S. J. Flockton, "Evolutionary algorithms with a coarse-to-fine function smoothing," in *Proc. IEEE Int. Conf. Evol. Comput.*, Perth, WA, Australia, 1995, pp. 657–62.
[33] L. A. Albert and D. E. Goldberg, "Efficient discretization scheduling in multiple dimensions," in *Proc. Genetic Evol. Comput. Conf.*, 2002, pp. 271–278.
[34] J.-H. Chen, D. E. Goldberg, S.-Y. Ho, and K. Sastry, "Fitness inheritance in multi-objective optimization," in *Proc. Genetic Evol. Comput. Conf.*, 2002, pp. 319–326.
[35] R. E. Smith, B. A. Dike, and S. A. Stegmann, "Fitness inheritance in genetic algorithms," in *Proc. ACM Symp. Appl. Comput.*, Nashville, TN, 1995, pp. 345–350.
[36] K. Sastry, D. E. Goldberg, and M. Pelikan, "Don't evaluate, inherit," in *Proc. Genetic Evol. Comput. Conf.*, 2001, pp. 551–558.
[37] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Norwell, MA: Kluwer, 2002.
[38] M. Pelikan and K. Sastry, "Fitness inheritance in the Bayesian optimization algorithm," in *Proc. Genetic Evol. Comput. Conf.*, Seattle, WA, 2004, pp. 48–59.
[39] A. Ochoa and M. R. O. Soto, "Partial evaluation of genetic algorithms," in *Proc. 1st Artif. Intell. Symp.*, Havana, Cuba, 1997, pp. 29–35.
[40] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-A survey," *IEEE Trans. Evol. Comput.*, vol. 9, pp. 303–317, 2005.
[41] D. A. Augusto and H. J. C. Barbosa, "Symbolic regression via genetic programming," in *Proc. VI Brazilian Symp. Neural Networks (SBRN'00)*, Rio de Janeiro, RJ, Brazil, 2000, pp. 173–173.
[42] J. Duffy and J. Engle-Warnick, "Using symbolic regression to infer strategies from experimental data," *Evol. Comput. Econ. Finance*, vol. 100, pp. 61–84, 2002.
[43] J. Eggermont and J. I. v. Hemert, "Stepwise adaptation of weights for symbolic regression with genetic programming," in *Proc. 12th Belgium/Netherlands Conf. Artif. Intell. (BNAIC'00)*, Kaatsheuvel, De Efteling, Holland, 2000, pp. 259–266.
[44] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.
[45] T. Soule and R. B. Heckendorn, "Function sets in genetic programming," in *Proc. Genetic Evol. Comput. Conf.*, San Francisco, CA, 2001, pp. 190–190.
[46] C. Ferreira, "Function finding and the creation of numerical constants in gene expression programming," in *Proc. 7th Online World Conf. Soft Comput. Industrial Applications*, 2002.
[47] B. Dolin, F. H. Bennett, III, and E. G. Rieffel, "Co-evolving an effective fitness sample: Experiments in symbolic regression and distributed robot control," in *Proc. 2002 ACM Symp. Appl. Comput.*, Madrid, Spain, 2002, pp. 553–559.
[48] 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 *Proc. 2002 Congr. Evol. Comput.*, 2002, pp. 1326–1331.
[49] M. Keijzer, "Improving symbolic regression with interval arithmetic and linear scaling," in *Proc. Genetic Programming, EuroGP'2003*, Essex, 2003, pp. 70–82.
[50] S. W. Mahfoud, *Niching Methods for Genetic Algorithms*. Urbana, IL: Univ. Illinois at Urbana–Champaign, 1995.
[51] A. J. Booker, J. E. Dennis, Jr, P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset, "A rigorous framework for optimization of expensive functions by surrogates," *Structural Opt.*, vol. 17, pp. 1–13, 1999.

Michael D. Schmidt received the B.Sc. degree in electrical and computer engineering and the M.E. degree in computer science from Cornell University, Ithaca, NY, in 2005 and 2006, respectively. He is currently working towards the Ph.D. degree at Cornell University.

His research interests focus on analytical modeling of complex systems, particularly, biological systems at the cellular and cellular network level, to understand how biological systems function and gain insight into why they have emerged in nature.

Hod Lipson (M'98) received the B.Sc. degree in mechanical engineering and the Ph.D. degree in mechanical engineering in computer-aided design and artificial intelligence in design from the Technion-Israel Institute of Technology, Haifa, Israel, in 1989 and 1998, respectively.

He is currently an Associate Professor at the Mechanical and Aerospace Engineering and Computing and Information Science Schools, Cornell University, Ithaca, NY. Prior to this appointment, he was a Postdoctoral Researcher at the Departmen of Computer Science, Brandeis University, and a Lecturer at the Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, where he conducted research in design automation. His research interests focus on computational methods for synthesizing complex systems out of elementary building blocks and the application of such methods to design automation and their implication to understanding the evolution of complexity in nature and in engineering.