Predicting Solution Rank to Improve Performance

Michael D. Schmidt
Computational Synthesis Lab
Cornell University
Ithaca, NY 14853
mds47@cornell.edu

Hod Lipson
Computational Synthesis Lab
Cornell University
Ithaca, NY 14853
hod.lipson@cornell.edu

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

Categories and Subject Descriptors
I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms
Algorithms, Design, Performance

Keywords
Fitness Prediction, Evolutionary Algorithms, Coevolution, Symbolic Regression

1. INTRODUCTION
In practice, many applications of evolutionary computation involve expensive fitness calculations [1, 2]. For example, some problems involve simulating the performance of evolved robotics or structures. Others commonly involve evaluating a solution over a large dataset.

One method to address the computational difficulty of fitness calculation is fitness modeling and approximation [3]. Fitness models are often coarse approximations of the full fitness calculation – for example, a coarse simulation, or subset of the dataset – chosen ahead of time to replace the full fitness function.

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

One general method to improve performance using fitness approximations in arbitrary applications is the Coevolution of Fitness Predictors algorithm [4-6]. Here, the concept of a fitness predictor is to estimate the exact fitness value of an individual with an extremely coarse and light-weight approximation. Instead of specifying the approximation ahead of time, fitness predictors are coevolved, optimizing their ability to estimate the exact fitnesses of the current solution population.

A surprising result from this method is that it can improve performance even with extremely coarse fitness approximations. For example, in the symbolic regression [7] problem, the fitness predictors can maintain an objective fitness gradient by evaluating solutions on as few as four data points in a data set of thousands of points and tens of variables [8]. In such extreme cases, the fitness approximations are almost certainly inaccurate, but still allow evolutionary progress on the objective fitness.

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

To test this idea, we use two implementations of the Coevolution of Fitness Predictors algorithm for symbolic regression [4]. The first is the standard fitness predictor algorithm which coevolves a small subset of the total training data to measure error. The second is identical, however fitness predictors are replaced with rank predictors, which are optimized to rank solutions, rather than model their fitness values. We then test the performance of these two algorithms on thousands of generated test problems and observe their differences over varying problem difficulties.

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

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

fitness landscape. Instead fitness predictors must be adapted throughout evolution.

Fitness approximations have been used in many applications of evolutionary computation. While most commonly used to reduce the computational cost of fitness evaluations [1, 2], fitness approximations can also be used for other tasks, such as smoothing rugged fitness landscapes [9], mapping discrete fitnesses to continuous values, and as a surrogate for human interactive evolution [10].

Many techniques exist for approximation fitness values and integrating them into the evolutionary search [3]. A few of the most important methods include fitness inheritance, fitness imitation, and partial evaluation. In fitness inheritance [11], 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 [12] builds a model of the fitness function based on the structure of the probabilistic model used in the algorithm [13]. In fitness imitation [14], 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 [15], the fitness of some individuals are calculated exactly, while others are modeled or inherited.

Sub-sampling of training data is also a common way to reduce the cost of fitness evaluation [16-18]. In many problems, fitness is calculated by evaluating individuals on training cases and combining the total error. With a sub-sample, only a fraction of the training data is evaluated in each generation.

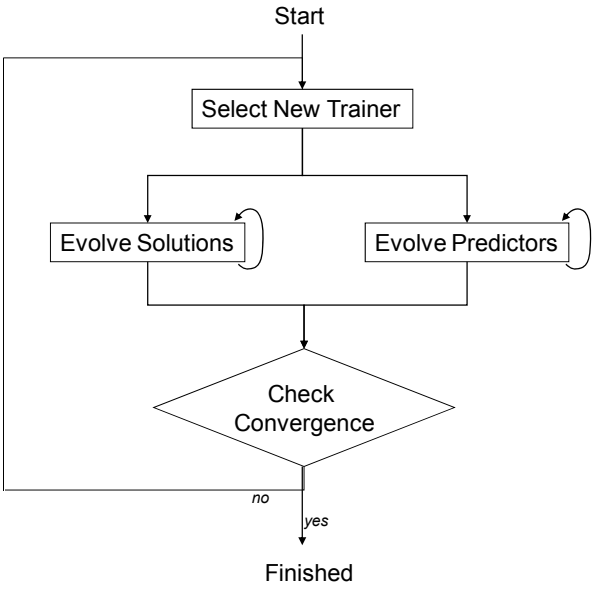

Figure 1. The algorithm structure of the rank predictor algorithm. Problem solutions and rank predictors are coevolved in parallel. Rank predictors optimize to accurately rank trainers – a set of solutions picked the solution population. The solution population is evolved using the best rank predictor, which is used for selection. The algorithm terminates when the solution population has converged to an epsilon error solution.

In this paper, we use a sub-sampling of training data for the predictor structure. For the fitness predictor, the sub-sample is optimized to match the fitness of the entire data set, while the rank predictor simply picks points that accurately rank the solutions. In both cases, the sample is optimized in a second coevolving population [4].

3. ALGORITHM

3.1 Fitness and Rank Predictors

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

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

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

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

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

3.2 Fitness and Rank Trainers

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

Fitness trainers are selected in order to help predictors optimize to the current solutions. To do this, the algorithm chooses a solution whose predicted rank or fitness has the least confidence. For example, we select the solution with the highest variance [19, 20] in predicted fitness, or highest variance in predicted rank, among the current rank or fitness predictors.

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

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

$$-\frac{1}{N}\sum_{i}\left|fitness(i)-prediction(i)\right|$$

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

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

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

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

3.3 Coevolution Algorithm

The coevolution algorithm [4] that we modify in this paper has three populations: Problem solutions, fitness predictors, and fitness trainers. As described earlier, fitness trainers are a set of solutions chosen to train the fitness and rank predictors on.

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 or rank predictors. The algorithm then evolves the solution population using the highest ranked fitness or rank predictor, and evolves the predictors using the fitness trainers. Finally, the highest-ranked individual is tested for convergence, and the algorithm completes if successful.

This basic algorithm structure is shown in Figure 1. Greater detail is provided in [4].

4. EXPERIMENTAL SETUP

In this section we detail our experimental methods to test the impact of using rank predictions rather than fitness approximations. We perform identical experiments on two algorithms: (1) the coevolved rank predictor algorithm, and (2) the coevolved fitness predictor algorithm [4].

We experiment on the Symbolic Regression problem because it is a ubiquitous and important problem in genetic programming [7]. Additionally, we can easily vary the problem complexity and the problem dimensionality.

Symbolic regression [7] is the problem of identifying the simplest equation [21] that most accurately fits a given set of data. Symbolic regression has a wide range of applications, such as prediction, classification, modeling, and system identification.

Recently, symbolic regression has been used to detect conserved quantities data representing physical laws of nature [22], infer the differential equations in dynamical systems [23].

4.1 Symbolic Regression

Symbolic regression [24] is a type of genetic program for searching the space of expressions computationally by minimizing various error metrics. Both the parameters and the form of the equation are subject to search. In symbolic regression, many initially random symbolic equations compete to model experimental data in the most parsimonious way. It forms new equations by recombining previous equations and probabilistically varying their sub-expressions. The algorithm retains equations that model the experimental data well while abandoning unpromising solutions. After an equation reaches a desired level of accuracy, the algorithm terminates, returning the most parsimonious equations that may correspond to the intrinsic mechanisms of the observed system.

In symbolic regression, the genotype or encoding represents symbolic expressions in computer memory. Often, the genotype is a binary tree of algebraic operations with numerical constants and symbolic variables at its leaves [25, 26]. Other encodings include acyclic graphs [8] and tree-adjunct grammars [27]. The fitness of a particular genotype (a candidate equation) is a numerical measure of how well it fits the data, such as the equation's correlation or squared-error with respect to the experimental data.

A point mutation can randomly change the type of the floating-point operation (for example, flipping an add operation to a

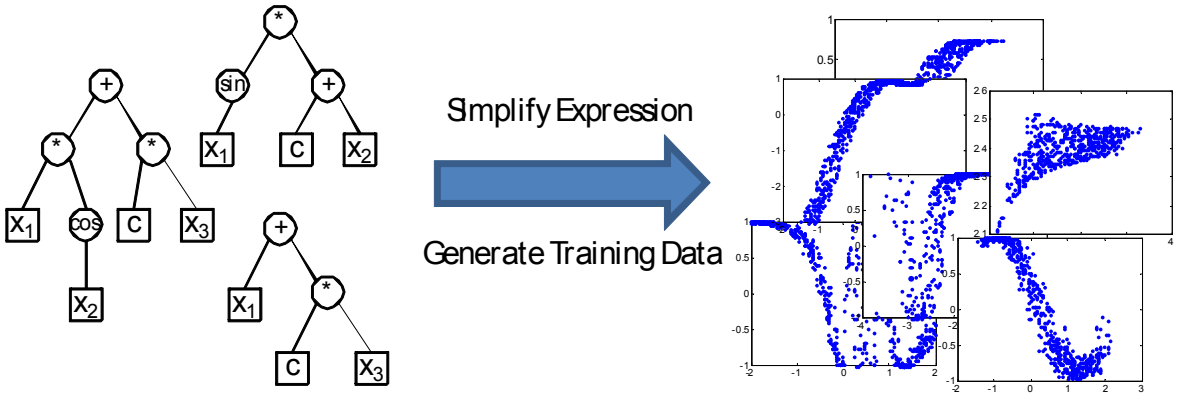

Figure 2. The generation of random test problems for symbolic regression. We start by picking a random number of inputs, between one and ten. We then generate a random equation using these inputs and simplify the equation before measuring its complexity (the number of nodes in the binary tree). We then generate a random training data set by sampling the input variables around the origin and evaluating the target equation on these data points. We then generate a validation data set in a similar fashion, but with a wider range around the origin to test if the solutions extrapolate to the exact solution.

multiply or an add to a system variable), or randomly change the parameter constant associated with that operation (if it is used). The crossover operation recombines two existing equations to form a new equation. To perform crossover, we select a random location in the genotype, and copy all operation and parameter values to the left of this point from the first parent and remaining operations and parameters to the right from the second parent.

4.2 Generating Test Problems

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

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

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

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

Many randomly generated equations may have compressible terms. For example, $f(x) = 4.211 + 0.93\ x^2 + 1.23$ is equivalent to $f(x) = 0.93\ x^2 + 5.441$. Therefore, we perform a symbolic simplification on the randomly generated equation in order to get an accurate measure of the target equations complexity. We measure complexity of the problem as the total nodes in the binary tree representation of the equation. For example, the complexity of the equation just above is 5.

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

Next, we randomly sample the input values of the equation 500 times to create a dataset. These variables are sampled from a normal distribution around the origin, with standard deviation of two. The equation is then evaluated on these variables in order to get the target output value. Several examples of training data are shown in Figure 2.

Finally, we also generate a separate validation data set of 500 points. The validation data set is created in the same fashion as the training data set, however the input variables are sampled with a standard deviation of three. By using a broader input sampling, we can use the validation dataset to test whether solutions extrapolate in their predictions to unseen data.

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

4.3 Measuring Performance

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

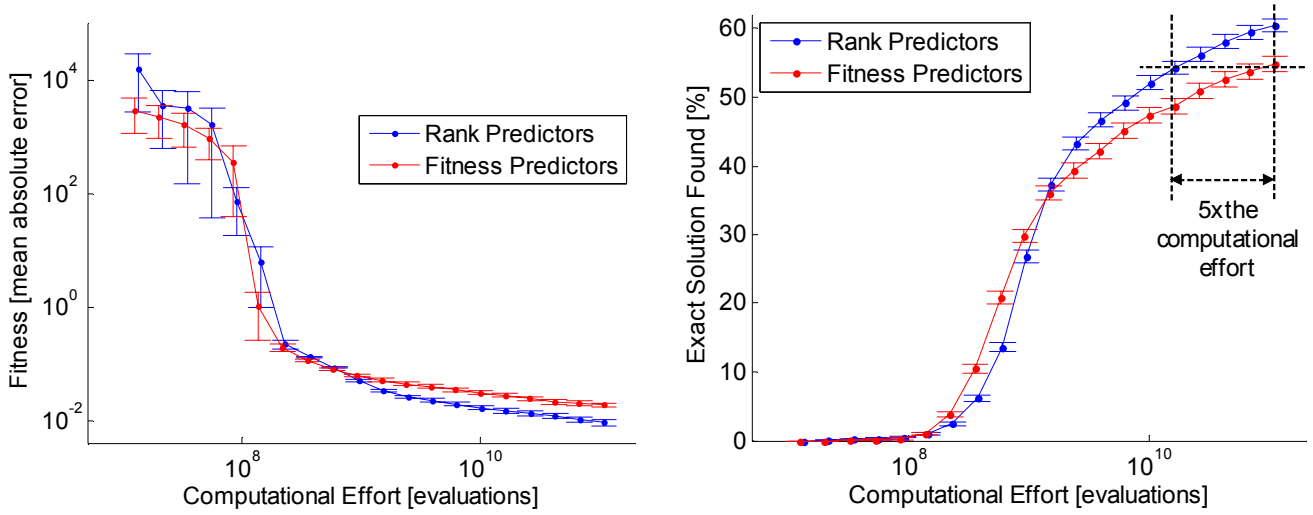

Figure 3. The fitness and convergence rate to the exact solution of the compared algorithms versus the total computational effort of the evolutionary search. The fitness is plotted (left) is the normalized mean absolute error on the validation data set. Fitness is normalized by the standard deviation of the output values. Convergence to the exact solution (right) is percent of the trials which reach epsilon error on the validation data set. The error bars indicate the standard error. Performance of the algorithm without prediction at all is several order of magnitude higher in computational effort and is not shown.

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

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

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

4.4 Algorithm Settings

We use the symbolic regression algorithm described in [4] as the basis for our implementation. We simply swap out the fitness criterion for the fitness predictor for the rank predictor criterion, described earlier.

We use the deterministic crowding selection method [28], with 1% mutation probability and 75% crossover probability. The encoding is an operation list acyclic graph with a maximum of 64 operations/nodes [8]. Single-point crossover exchanges operations in the genome at a random split. The operation set contains addition, subtraction, multiply, divide, sine, and cosine operations.

We limit the size of the equation to narrow our search to human-interpretable equations (equations we could fit on a piece of paper). We allowed a maximum of 64 nodes, each possibly representing six types of mathematical operations, 1 to 10 variables, or a parameter constant. Ignoring the infinite parameter space, we are effectively searching a space of roughly over 10^{54} parameterized genotypes.

5. EXPERIMENTAL RESULTS

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

5.1 Fitness and Convergence

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

The fitnesses plotted in Figure 3 show both algorithms have similar trends on the randomly generated test problems, suggesting that the algorithms experience similar optima during their searches. Despite this, we see a clear difference in the fitness performance over time, with rank predictors achieving lower error.

This may also reflect the difference in convergences to the exact problem solution, also plotted in Figure 3. Here we notice that the fitness predictor algorithm begins finding exact solutions slightly sooner than the rank predictor algorithm. However, it is quickly overcome by the rank predictor algorithm.

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

5.2 Computational Effort

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

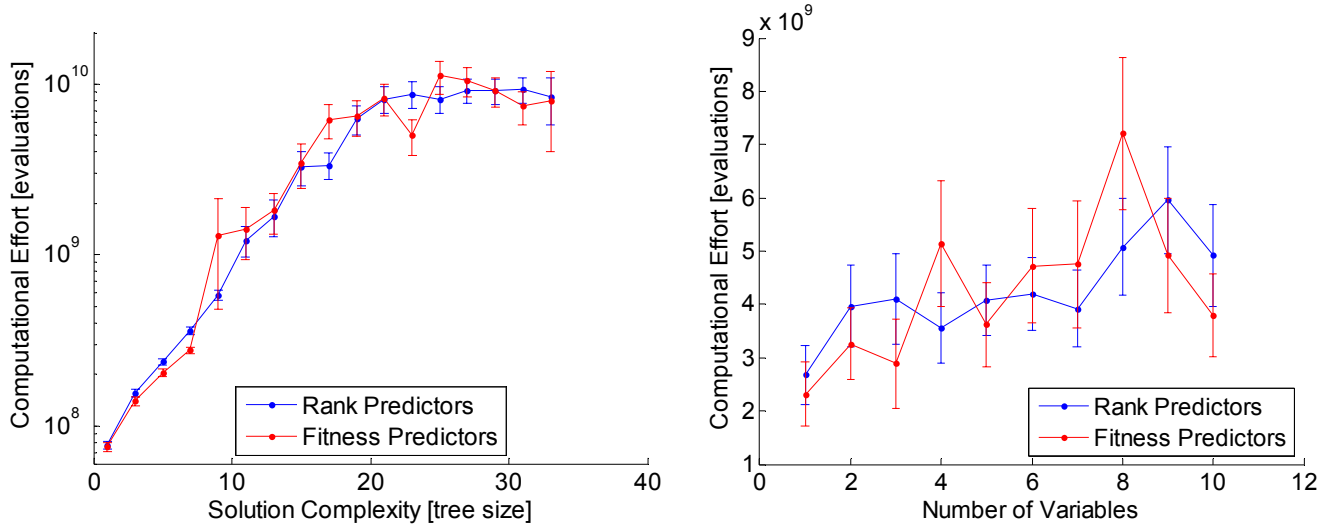

Figure 4. The computational effort required when the exact solution was found versus the target equation complexity (left) and the number of variables in the dataset (right). Each algorithm found the exact solution with different frequencies; these plots show the computation effort for when the algorithms did find the exact solution. The error bars indicate the standard error.

In response to increasing target solution complexity, shown in Figure 4, both algorithms show very similar trends. We do see a small difference, where the fitness predictor algorithm tended to find the exact solution slightly faster for the simplest problems. At higher complexities, the difference is less noticeable, however rank predictors tended to require slightly less effort at most higher complexities than the fitness predictors.

There is a similar trend found in the computational effort to find the exact solution versus the number of variables in the problem datasets (Figure 4). Computational effort tended to increase with dimensionality for both algorithms. Again, fitness predictors tended to require slightly less effort on average for the lower dimensions.

5.3 Solution Bloat

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

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

The bloat results (Figure 5) show that both algorithms begin with highly bloated solutions, which decrease over the search toward the target solution on average.

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

6. DISCUSSION

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

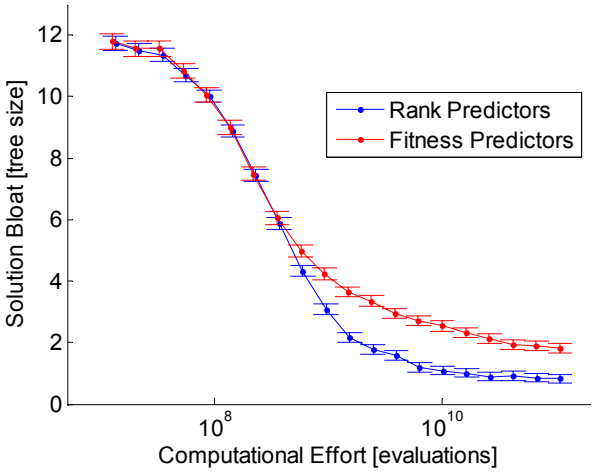

Figure 5. Solution bloat over the course of the evolutionary search. Solution bloat is defined as the binary tree size of the best individual in the population minus the binary tree size of the target solution. The error bars indicate the standard error.

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

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

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

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

7. CONCLUSION

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

We compared the difference between optimizing modeled fitness values and optimizing solution rankings using a coevolutionary algorithm which optimizes either fitness predictors or rank predictors with the evolving problem solutions. We tested both methods on the symbolic regression problem using thousands of test problems, varying in problem complexity and number of variables.

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

8. ACKNOWLEDGMENTS

This research is supported by the U.S. National Science Foundation (NSF) Graduate Research Fellowship Program, NSF Grant ECCS 0941561 on Cyber-enabled Discovery and Innovation (CDI), and the U.S. Defense Threat Reduction Agency (DTRA) grant HDTRA 1-09-1-0013.

9. 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 *Proceedings of the 2001 Congress on Evolutionary Computation*, 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] M. D. Schmidt and H. Lipson, "Coevolution of Fitness Predictors," *IEEE Transactions on Evolutionary Computation,* vol. 12, pp. 736-749, Dec 2008.

[5] 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. 113-130.

[6] 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.

[7] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*: MIT Press, 1992.

[8] M. Schmidt and H. Lipson, "Comparison of tree and graph encodings as function of problem complexity," in *Proceedings of the Genetic and Evolutionary Computation Conference*, London, 2007, pp. 1674--1679.

[9] D. Yang and S. J. Flockton, "Evolutionary algorithms with a coarse-to-fine function smoothing," in *1995 IEEE International Conference on Evolutionary Computation*, Perth, WA, Australia, 1995, pp. 657-62.

[10] H. Takagi, "Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evaluation," *Proceedings of the IEEE,* vol. 89, pp. 1275--1296, 2001.

[11] R. E. Smith, B. A. Dike, and S. A. Stegmann, "Fitness inheritance in genetic algorithms," in *Proceedings of the ACM Symposium on Applied Computing*, Nashville, TN, USA, 1995, pp. 345-350.

[12] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*: Kluwer Academic Publishers, 2002.

[13] M. Pelikan and K. Sastry, "Fitness inheritance in the Bayesian optimization algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*, Seattle, WA, USA, 2004, pp. 48-59.

[14] Y. Jin and B. Sendhoff, "Reducing Fitness Evaluations Using Clustering Techniques and Neural Network Ensembles," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2004, pp. 688-699.

[15] A. Ochoa and M. R. Soto Ortiz, "Partial evaluation of genetic algorithms," in *1st Artificial Intelligence Symposium*, Havana, Cuba, 1997, pp. 29-35.

[16] A. Teller and D. Andre, "Automatically Choosing the Number of Fitness Cases: The Rational Allocation of Trials," in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, 1997, pp. 321--328.

[17] L. A. Albert and D. E. Goldberg, "Efficient Discretization Scheduling In Multiple Dimensions," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2002, pp. 271-278.

[18] L. Pagie and P. Hogeweg, "Evolutionary Consequences of Coevolving Targets," *Evolutionary Computation,* vol. 5, pp. 401--418, 1997.

[19] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *IEEE Transactions on Evolutionary Computation,* vol. 9, pp. 303-317, 2005.

[20] J. C. Bongard and H. Lipson, "Nonlinear System Identification Using Coevolution of Models and Tests," *IEEE Transactions on Evolutionary Computation,* vol. 9, pp. 361-384, 2005.

[21] P. Grünwald, "Model selection based on minimum description length," *J. Math. Psychol.,* vol. 44, pp. 133-152, 2000.

[22] M. Schmidt and H. Lipson, "Distilling Free-Form Natural Laws from Experimental Data," *Science,* vol. 324, pp. 81-85, April 3, 2009 2009.

[23] J. Bongard and H. Lipson, "Automated reverse engineering of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences,* vol. 104, pp. 9943-9948, June 12, 2007 2007.

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

[25] M. Ben, J. W. Mark, and W. B. Geoffrey, "Using a Tree Structured Genetic Algorithm to Perform Symbolic Regression," in *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*. vol. 414, A. M. S. Zalzala, Ed. Sheffield, UK: IEE London, UK, 1995, pp. 487--492.

[26] D. Edwin and B. P. Jordan, "Multi-Objective Methods for Tree Size Control," in *Genetic Programming and Evolvable Machines*. vol. 4, 2003, pp. 211--233.

[27] X. H. Nguyen, R. I. McKay, and D. L. Essam, "Solving the Symbolic Regression Problem with Tree-Adjunct Grammar Guided Genetic Programming: The Comparative Results," in *The Australian Journal of Intelligent Information Processing Systems*. vol. 7, 2001, pp. 114--121.

[28] S. W. Mahfoud, "Niching methods for genetic algorithms," University of Illinois at Urbana-Champaign, 1995.