Trainer Selection Strategies for Coevolving Rank Predictors

Daniel L. Ly[1] and Hod Lipson[1,2]
[1] Department of Mechanical and Aerospace Engineering
[2] Department of Computing and Information Sciences
Cornell University
Ithaca, New York 14853
{dll73, hod.lipson}@cornell.edu

***Abstract*—Despite the range of applications and successes of evolutionary algorithms, expensive fitness computations often form a critical performance bottleneck. A preferred method of reducing the computational overhead is to coevolve rank predictors, providing a coarse and lightweight fitness approximation that has proven to drastically increase performance. However, the majority of previous work on rank predictor coevolution focused solely on improving the predictor heuristics while strategies to select the equally important trainer population is often an afterthought. Four different strategies are presented and benchmarked on a symbolic regression problem using hundreds of test problems with varying complexities. Of the four strategies, updating the trainer population with the solution of the highest rank variance is found to be significantly superior, resulting in a four to ten fold reduction in computational effort for similar convergence rates over the remaining strategies.**

I. Introduction

Evolutionary algorithms are popular, stochastic, population-based, heuristic algorithms that have been successfully applied to a wide range of optimization and search problems [1]. Despite the variety of applications, a common criticism and prohibitive limitation in practice stems from the computationally heavy demands of these algorithms [2]. Often, the primary culprit in the computational requirements arises from calculating fitnesses [3]. Evolutionary algorithms operate by incrementally improving a population via iteratively selecting the most fit individuals. In many applications, determining the fitness of a single individual requires repeatedly evaluating a local metric over large data sets [4].

A general approach to alleviate the computational requirements involves modelling and approximating fitnesses [2]. Rather than calculating expensive fitnesses across large data sets, a coarser and lightweight approximation can often be substituted. A particularly effective method to improve performance is by coevolving predictors [5], [6]. Instead of using the complete data set, the fitness is measured only on a dynamic subset of the data. The members of this subset, called *predictors*, are coevolved simultaneously based on the solution population, allowing for evolutionary progress through direct competition. Surprisingly, large data sets can be sufficiently modelled by a subset that are orders of magnitude smaller, resulting in drastic improvements in performance. In a symbolic regression problem, data sets that consist of thousands of points and tens of variables can be effectively replaced by a dynamic selection of just four points [5].

The key to this coevolution technique relies on the systematic method of evolving predictors. First, predictors were rewarded on their ability to predict the exact fitness [5]. However, it was shown that rewarding predictors based on their ability to rank solutions, rather than using fitness measurements directly, proved to be a far superior method [6]. Although this extremely coarse approximation almost certainly provides inaccurate fitnesses, it is still capable of promoting evolutionary progress and, in fact, results in a 5-fold performance improvement over fitness predictors.

However, an often overlooked but equally essential component of these predictor coevolution algorithms is the *trainer* selection strategy. The trainer population is a "hall of fame" of solution individuals which are used to evolve the predictors, and thus are integral in the emergent behaviour of the system. Despite their critical role, strategies for selecting trainers have not been thoroughly investigated and compared.

This paper presents four different trainer selection strategies for coevolving rank predictors. The performance of these strategies are tested and compared on hundreds of randomly generated test problems over a range of difficulties.

The rest of the paper is organized as follows: Section II discusses the related work in fitness approximation and coevolution. Section III describes the rank prediction algorithm and the four trainer selection strategies. The experimental setup is detailed in Section IV and the results are presented and analyzed in Section V. The paper is concluded in Section VI.

II. Related Work

This work draws from two distinct fields: fitness modelling and competitive coevolutionary algorithms. Fitness modelling is a well understood technique of using predefined models or coarse simulations to approximate fitness calculations in evolutionary algorithms with the primary goal of reducing the computational overhead [2]. Although they are most commonly used to increase the performance of fitness evaluations, fitness approximations have also found alternative uses in other fields such as approximating fitnesses where no explicit model exists [7], dealing with noisy fitness functions [8] and modelling multi-modal fitness landscapes [9].

There are several different techniques for approximating fitness values as a part of the evolutionary algorithm. The two most prominent methods are fitness inheritance and fitness imitation. In fitness inheritance, fitness computations are saved by estimating the fitness of offspring individuals based on their parents during the recombination process [10]. This approach builds a model of the fitness function based on the probabilistic components of the algorithm [11], [12]. The second class is fitness imitation, which is built on the assumption that similar genotypes, or solution representations, will share similar phenotypes, or solution fitnesses. The first step is to cluster individuals into groups based on genotypic distance. Only the fitness of a representative individual of each cluster is computed directly and it is assigned to the other individuals in that cluster [13], [14].

In contrast, fitness predictors are a class of fitness modelling approaches that are so coarse that they are unable to approximate the entire fitness landscape. Instead, predictors are coevolved with the solution population to cater to the evolutionary process via direct competition. Predictors reduce the computational overhead of fitness evaluations by subsampling the training data [5], [15], [16], [17]. Since the fitness metric of many problems is often defined as a sum of local errors, this subsampling approach ignores the majority of the data with the exception of points described by the systematically evolved predictors.

The second area of relevant research consists of coevolutionary algorithms – heuristic approaches which use multiple populations with interdependent fitnesses. In competitive coevolution, the populations reciprocally drive each other to increasing levels of performance and complexity [18]. However, for non-trivial problems, a persisting challenge arises from the computational complexity of mixed fitness assessment between large populations. A popular solution to this issue is the "hall of fame" technique, where a subset of good individuals from prior generations are saved and used for comparison [18], [19].

In this paper, rank predictors are used in a competitive coevolutionary algorithm to subsample the training data and are rewarded for their ability to accurately rank solutions [6]. Previous work in this field focused on developing heuristics for evolving predictors, while this work investigates trainer selection strategies. Trainers act as a "hall of fame" for the solution population and the strategies are selection methods to populate the hall.

III. Rank Prediction Algorithm

This section begins with an overview of the predictor coevolution framework followed by an illustrative example of rank predictors in a regression problem. Four trainer selection strategies are then presented and discussed.

A. Predictor Coevolution Framework

A traditional evolutionary algorithm can be viewed as an optimization problem to find the most fit *solution* to a given data set. Formally, given a space of candidate solutions, $\mathbb{S}$, the optimal solution, s^* is defined as:

$$s^* = \underset{s \in \mathbb{S}}{\operatorname{argmax}} F(s; D) \tag{1}$$

where $F(s; D)$ is the *solution fitness* of the candidate solution s evaluated on the training data set $D = \{d_1, \ldots, d_N\}$. When the solution fitness is evaluated on the complete data set, the result is called the *exact fitness*. Evolutionary techniques maintain a population of candidate solutions, $\mathcal{S}$, and the algorithm proceeds by selecting the most fit individuals and recombines them iteratively with the goal of producing increasingly fit solutions in each generation.

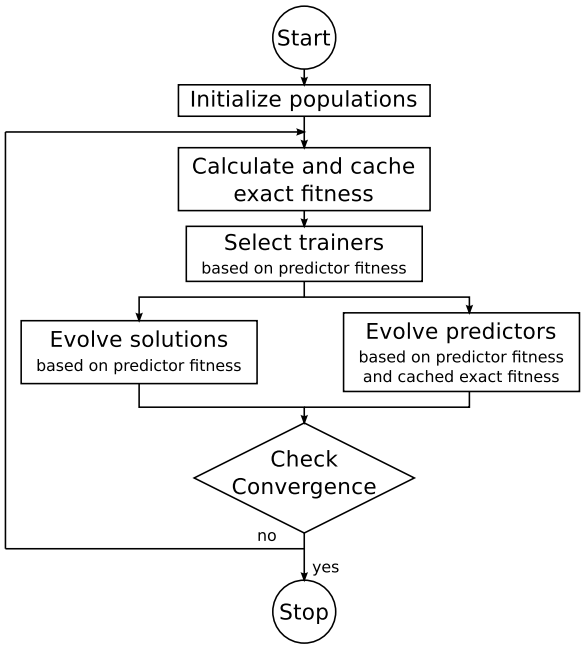


Fig. 1. A flowchart of the predictor coevolution process.

In the predictor coevolution algorithms, two additional populations are introduced – *predictors* and *trainers* – which results in several key changes to the underlying optimization problem. First, solutions are no longer evolved to optimize exact fitnesses. Rather, the solution fitness is evaluated only on the current best predictor; the resulting fitness is called the *predictor fitness*. Since a predictor has drastically fewer points than the complete data set, it is significantly faster to calculate predictor fitnesses than exact fitnesses, resulting in large performance gains.

Next, the trainer population, $\mathcal{T}$, is a "hall of fame" of solutions selected according to a *trainer strategy*, $T(\cdot)$, to aid in the evolution of predictors. Thus, the trainer strategy's role is to select solutions from the population based on the current predictor population.

The final population is the predictor population, $\mathcal{P}$. Each predictor is a small subset of the complete data set, $p = \{d_1, \ldots, d_K\} \subset D$. Predictors are coevolved with the solutions based on the *predictor fitness*, $R(\cdot)$, which fundamentally is a comparison between predictor fitnesses and

Exact fitness	The solution fitness measured on the complete data set
Predictor population	The secondary population used for fitness approximation and competitive coevolution
Predictor fitness	An objective function that prescribes the optimality of the predictor population
Rank predictors	Predictors whose fitness is a function of their ability to rank solutions
Solution population	The primary optimization population
Solution fitness	An objective function that prescribes the optimality of the solution population
Trainer population	A tertiary population of selected solutions used to train predictors
Trainer strategy	An algorithm to select trainers from the solution population

TABLE I
GLOSSARY OF TERMS

exact fitnesses evaluated using the trainer population. Thus, the evolutionary process rewards predictors that can accurately predict the behaviour of the trainer population. The three evolutionary rules are formally defined as follows:

$$s^* = \underset{s \in \mathbb{S}}{\operatorname{argmax}} F(s; p^*) \tag{2}$$

$$t^* = \underset{s \in \mathcal{S}}{\operatorname{argmax}} T(F(s; p)) \quad , \forall p \in \mathcal{P} \tag{3}$$

$$p^* = \underset{p \in D}{\operatorname{argmax}} R\left(F(t; p), F(t; D)\right) \quad , \forall t \in \mathcal{T} \tag{4}$$

The predictor coevolution process is summarized in Fig. 1 and a glossary of terms in included in Table I. This framework provides several benefits compared to a traditional implementation of evolutionary algorithms. The primary benefit is that the expensive exact fitness calculations are replaced with the light-weight predictor fitness calculations. In predictor coevolution, exact fitnesses are only computed when the trainer population is updated; these values are calculated once and cached for the predictor evolution until the next update. Other benefits include faster convergence rates and reduced bloat [5].

B. Rank Prediction Example

This subsection illustrates the theoretical predictor framework via a regression example and depicts how rank predictors are implemented. The optimization problem is: given the data set shown in Fig. 2, find the parameters of a cubic function that minimizes the normalized mean absolute error. For a data point $d_n = (x_n, y_n)$, the following relationship is assumed:

$$y = \sum_{i=0}^{4} w_i x^i \tag{5}$$

As a result, each solution represents a collection of weight parameters, $s = \{w_0, \ldots w_4\}$, and the optimization problem is to find the parameter values that produce the highest fitness.

Let $\hat{y}_n(s, x_n)$ be the prediction from a candidate solution s on data point n, then the solution fitness is calculated as the normalized mean absolute error:

$$F(s; p) = -\frac{1}{K} \sum_{k=1}^{K} \frac{|\hat{y}_k - y_k|}{\sigma_y} \tag{6}$$

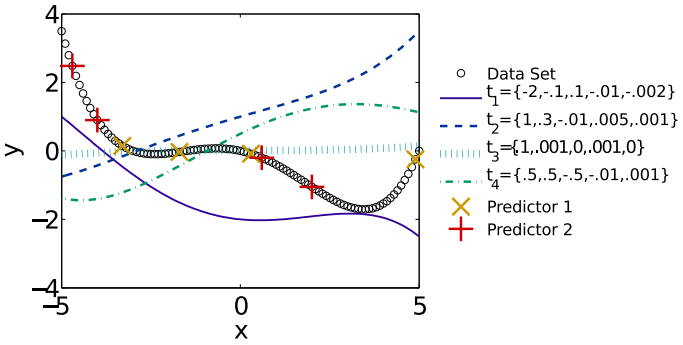


Fig. 2. The data set, trainers and predictors for the rank prediction example. The data set consists of 101 points generated from a noiseless cubic function. Four trainers, along with their parameter values, are provided. Each predictor consists of only four points from the data set.

Trainer		Predictor 1			Predictor 2		
Rank	Exact fitness	Rank	Pred. Fitness	Rank Error	Rank	Pred. Fitness	Rank Error
t_3	-0.78	t_3	-0.17	0	t_3	-1.20	0
t_1	-1.18	t_4	-0.97	1	t_1	-1.29	0
t_4	-1.71	t_2	-1.40	1	t_2	-2.13	1
t_2	-1.95	t_1	-1.55	2	t_4	-2.37	1
		Rank Fitness:		-4	Rank Fitness:		-2

TABLE II
EXAMPLE RANK FITNESS CALCULATION (EQ. 7)

where σ_y is the standard deviation of y. Note that the solutions are evolved using only the predictor fitness.

Trainers are a dynamic collection of solutions used to evolve predictors. In this example, an arbitrary set of trainers are provided (Fig. 2). However, strategies to select trainers from the solution population are described in Section III-C.

Predictors are a subset of the data. Two examples of predictors are shown in Fig. 2. In rank prediction, the exact and predictor fitnesses are calculated and used to rank the trainers. The rank fitness is then calculated as the Manhattan distance between the ranks of each trainer:

$$R\left(F(t; p), F(t; D)\right) = -\sum_{i=1}^{|\mathcal{T}|} \Big|\text{rank}[F(t_i; p)] - \text{rank}[F(t_i; D)]\Big| \tag{7}$$

This rank fitness metric rewards predictors that can correctly predict the same ranking of trainers as produced by the exact fitnesses. This metric is similar to the previous metric described in [6] but provides a smoother fitness landscape by using a distance-based metric rather than a Boolean-based metric. An example of rank fitness calculations is shown in Table II. Note that the predictor fitnesses are not used directly and only the ranking is used to determine rank fitnesses. In this example, all the values of predictor 1 lie near the zeros of the cubic function and thus it has limited ranking capabilities. In contrast, predictor 2 highlights some of the more relevant features of the data set and is superior to predictor 1 as it obtains a higher rank fitness for the given trainer population.

C. Trainer Selection

Trainers are a population of solutions which are used to evolve rank predictors, and consequently, play a critical role in the emergent behaviour of the coevolved system. A diverse and meaningful trainer population encourages effective predictors while a monotonic trainer population provides no gradient for predictor evolution and greatly stifles the evolutionary progress. However, the optimal strategy to select trainers from the solution population is not obvious and requires further investigation.

A series of heuristic, rule-based strategies are presented for trainer selection. A primary criteria in these trainer selection strategies is a low computational overhead since the ultimate goal is to increase the performance of evolutionary algorithms. Four different strategies, representing a range of concepts and complexities, are investigated:

1) **Random selection** [*RAN*] – This strategy is the simplest and most naive method to populate the trainer population, providing a baseline for comparing strategies. Given a solution population, select solutions randomly with a uniform distribution:

$$T_{\text{RAN}}\left(F(s;p)\right) = U(1, |\mathcal{S}|) \tag{8}$$

2) **Rank variance** [*VAR*] – A more methodological approach to selecting solutions is to pick solutions with the highest variance in rank. Given a predictor population, rank all of the solutions for each predictor and pick the solutions that have the greatest discrepancies in the rank:

$$T_{\text{VAR}}\left(F(s;p)\right) = -\sum_{i=1}^{|\mathcal{P}|}(\text{rank}[F(s;p_i)] - \langle\text{rank}[F(s;p_i)]\rangle)^2 \tag{9}$$

where $\langle\cdot\rangle$ denotes the mean value. This strategy is closely related to previous work on using coevolution to guide the search by exploring areas of least confidence [20], [21]. Variance provides an inverse measure of rank prediction confidence, and thus, selecting solutions with the highest variance may provide a better guide to the predictor evolution.

3) **Rank variance with memory** [*MEM*] – This strategy builds upon the *VAR* approach, except the trainer population is treated with memory. Rather than completely replacing the entire trainer population in a single update, only a single trainer is switched. The trainers are kept in a queue data structure. In each update, the oldest trainer is removed and is replaced by a new trainer of the highest variance in the current population. The motivation of this approach is to minimize drastic changes to the trainer population, since this may adversely affect the predictor evolution.

4) **Rank-variance/exact-fitness Pareto optimal** [*RFP*] – The final trainer selection strategy is the most complex approach, which again leverages the concepts from the *VAR* strategy. Although fitness rankings may vary widely based on the predictor, the rank of each solution is not completely independent and there are noticeable correlations and similarities amongst the rankings. By the nature of variance measurements, the solutions with the highest variance are likely those around the median rank. Thus, strategies that rely solely on variance are often comprised of trainers of median rank.

While this distribution of trainers is not an issue by itself, it must be analyzed within the context of an important observation: predictor coevolution does not guarantee elitism on exact fitnesses even with elitist selection methods for solution evolution. Elitism is the property that the most fit solution must always exist in the next generation. However, since solutions are evolved only using the predictors, elitism can only be guaranteed with respect to predictor fitnesses, not exact fitnesses.

This strategy attempts to imitate elitism by ensuring that solutions with high exact fitnesses are not underrepresented in the trainer population. This attempts to balance the information content of the population, or solutions of high variance, and the significance of the population, or solutions of high exact fitness.

Similar to the previous approaches, given a solution population, the rank variance is computed. However, trainers are prioritized based on their exact fitness and whether they are rank-variance/exact-fitness Pareto optimal – that is, solutions that provide the highest variance for a given exact fitness. However, since the number of Pareto optimal solutions may not equal the size of the trainer population, trainers are selected according to the following priorities:

i. The solution with the highest exact fitness
ii. Solutions that are rank-variance/exact-fitness Pareto optimal, beginning with the highest variance
iii. Solutions that are not rank-variance/exact-fitness Pareto optimal, beginning with the highest variance

Note that, despite the design considerations, this strategy does not guarantee elitism on exact fitnesses. Since the trainer population is only a subset of the solution population, a perfect predictor can only ensure that the trainers are ranked accurately relative to each other and has no oversight in how the trainers rank relative to the solution population. However, this strategy does bias the trainer population to ensure a diversity that both high rank variance solutions and high exact fitness solutions are well represented which may provide an edge in the emergent coevolution behaviour.

IV. Experimental Setup

This section describes the experimental methods used to evaluate and compare the four trainer strategies. A large, randomly generated test suite on the popular symbolic regression problem is used as the basis of comparison.

A. Symbolic Regression

Symbolic regression is a type of genetic programming optimization problem that searches the realm of symbolic

expressions to minimize a specific error metric over a given data set [22]. It is an important and ubiquitous problem in genetic programming with many applications in prediction, modelling and system identification. Recent advancements in this field have extended the capabilities of symbolic regression to finding conserved, physical laws of nature [23] and inferring differential equations in dynamical systems [24]. Symbolic regression is also a good test platform because the problem difficulty is readily varied by continuous parameters such as dimensionality of the data and complexity in the problem size.

Symbolic regression is a regression problem similar to the example discussed in Section III-B. However, rather than assuming an expression and optimizing parameters within that equation, the underlying structure of the expression is a primary component of the search problem. By using data structures that can represent any arbitrary expression, the regression problem includes finding a symbolic representation of the expression that best represents the data set. This problem is useful as it allows regression to be applied to problems where no prior assumptions can be made regarding the structure of the data source.

In symbolic regression, the genotype, or solution encoding, is capable of representing an arbitrary symbolic expression. A common representation consists of binary trees with algebraic operations on symbolic variables [25], [26], while other work have investigated alternative representations including acyclic graphs [27] and tree-adjunct grammars [28]. These representations are amenable to evolutionary processes such as mutation and recombination. Stochastic point mutations are applied in a variety of ways – for example, point mutations replace an add operation with a multiply operation or exchange one variable for a constant. Crossover recombination, which combines two existing parent expressions into a new offspring expression, can be implemented by selecting random locations in the genotype and switching subexpressions.

The phenotype, or output behaviour, of a candidate solution is determined by resolving the expression computationally. Provided with a collection of inputs from the data set, the expression is evaluated at each point by substituting the inputs and computing each subexpression iteratively. The output can then be inserted into any standard measure, such as absolute error or correlation measures, to determine the solution fitness.

Symbolic regression was chosen as test platform for predictor coevolution for two reasons. First, the majority of fitness metrics are often local measures, which are often computed as sums of evaluations on individual points. The local independence in the fitness metrics makes selecting predictor subsamples trivial – this is not the case for problems that require fitnesses based on time series or other holistic measures. Next, there is a significant body of work regarding the development of coevolution for symbolic regression, which provides a basis of comparison [5], [6].

B. Randomly Generated Test Suite

The trainer selection strategies were evaluated and compared on 256 data sets of various difficulties. To provide a fair comparison, random expressions were generated based on parameters that were swept uniformly in difficulty. The expression was then evaluated on random input data to complete the construction of the data set.

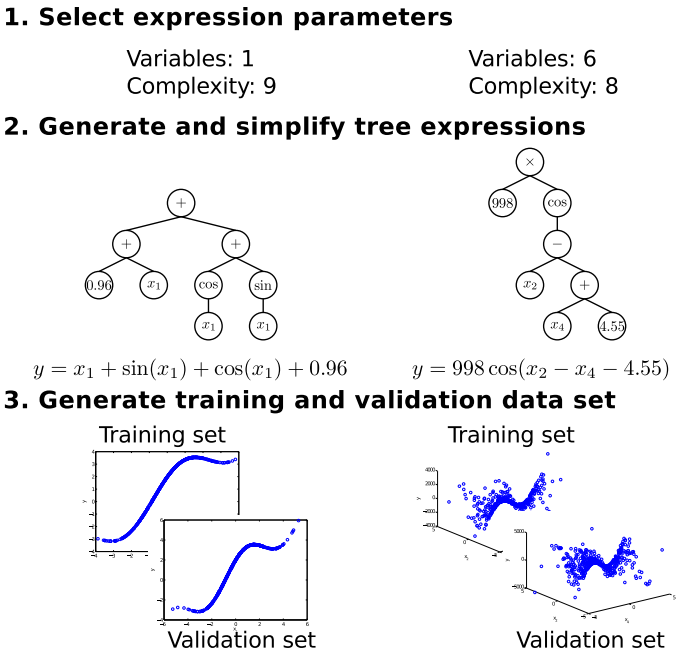


Fig. 3. Randomly generated expressions of varying complexities.

The first step in generating random expressions is to determine the expression parameters. There are two parameters that define the difficulty of the symbolic regression problem: the dimensionality of the data, or the number of variables in the data set, and the complexity of the target function, measured as the total number of nodes in the expression's binary tree representation. For this test suite, the dimensionality of the problem was swept from 1 to 8 variables, while the expression complexity was swept from 1 to 32 nodes.

Next, a random tree was generated by continually adding subtrees, using any of the available variables, until the target complexity was met. Since many expressions have a compressible form, a symbolic simplification was required to ensure that the measure of difficulty is accurate and consistent. For example, the expression $y = 8.4 + x_2 - 2.3 - x_2 + 1.7x_1$ can be simplified to lowest terms as $y = 6.1 + 1.7x_1$, resulting in a complexity of four nodes and dimensionality two variables (although x_2 is not in the expression, it is still part of data set and thus part of the search problem).

The final step is to randomly generate data points. A training data set for each expression set was constructed of 512 points. Each variable was sampled from an independent Gaussian distribution centered around the origin with a standard deviation of two. The expression was evaluated on the inputs and the resulting value was recorded as the target output. Next, a separate validation data set was also generated, consisting of 512 points as well. This data set is computed in the same fashion as the training data set, with the exception that the

standard deviation of the Gaussian distributions was expanded to three. By allowing for a broader range of inputs, the validation set tests whether solutions are able to extrapolate into unseen data.

In summary, the data set consisted of 256 randomly generated expressions ranging from 1 to 8 variables and 1 to 32 nodes. Each data set has a training and validation set of 512 points, generated from Gaussian distributions of two and three standard deviations, respectively. The data was generated without noise for convenience and ease of analysis; however, previous work indicates that symbolic regression is capable of inferring models from noisy data [23], [24]. Furthermore, as the trainer population is relatively isolated from the data, the effect of noisy data on the trainer strategy is expected to be minimal. This entire process is shown in Fig. 3.

C. Experimental Parameters

For the symbolic regression problem, the following evolutionary parameters were used. Mutation and crossover was done stochastically with a 1% mutation probability and 50% crossover probability. Point mutations were used and crossover was executed at a single point which was determined stochastically. Selection was done via the deterministic crowding method, which was used to minimize premature convergence on local optima [29]. A greedy, hill-climbing subalgorithm was used to optimize constant parameter values.

Expressions used a binary tree encoding with a maximum size of 64 nodes, which was chosen as the limit of human-interpretable expressions or equations that fit on a single piece of paper. The operation set contained six types: addition, subtraction, multiplication, division, sine and cosine operations. With tree sizes of 64 nodes, six operators, eight variables and parameter constants, the symbolic regression is effectively searching an expression space on the scale of approximately 10^{27} binary trees, in addition to the infinite parameter space of real-valued constants.

The solution population consisted of 128 individuals. The predictor population had eight individuals and each predictor was a subset of 8 points from the training data set. The trainer population consisted of 8 solutions and was updated every 100 generations. All of the experimental parameters were chosen based on previous work [6], and exhaustive parameter optimization was not conducted.

D. Implementation Details

Each trainer selection strategy was evaluated on the data set of 256 expressions in two distinct runs, resulting in a total of 512 experiments for each strategy. The experiments used the same code base, with the exception of the strategies, which were swapped accordingly. Furthermore, to ensure consistency between the strategies within each run, the random number generators in the coevolved system were seeded with the same series of predetermined values. This ensured that they all began with the same random initial populations and equivalent results are obtained if the strategies made the same decisions. Thus, the coevolution algorithm was still stochastically different between runs, but identical between strategies in the same run.

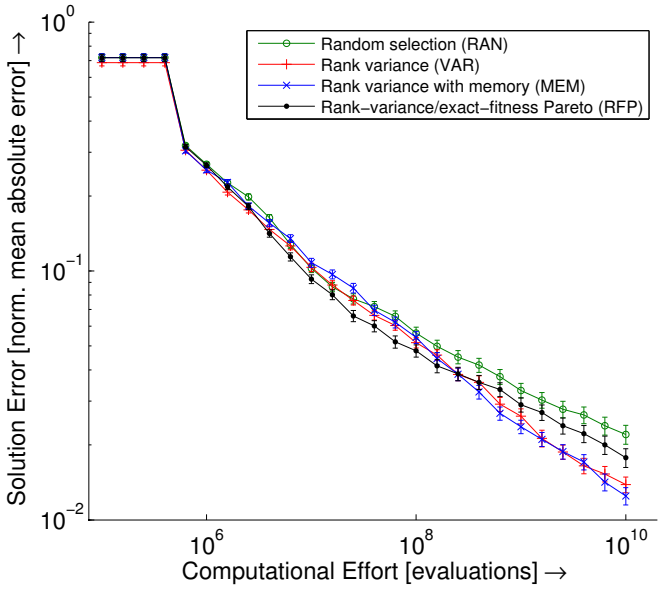


Fig. 4. Normalized mean absolute error of the trainer selection strategies averaged across the test suite versus computational error of the coevolution algorithm. The error bars indicate standard error of the mean.

Two metrics are of interest: exact fitness and convergence percentage. The fitness metric used in the problem was normalized mean absolute error (Eq. 6). Normalized fitness was used since it allowed comparisons between different data sets and simplified the detection of convergence. Convergence is defined as the algorithm finding the exact target expression. Since the data sets were generated without noise, the algorithm was considered to have found the exact target expression if any of the solutions computed an exact fitness of less than 10^{-3} normalized mean absolute error on the validation data set. The evolved solutions were subsequently confirmed to be symbolically identical to the target expression.

Throughout the search, the best solutions were logged, along with the exact fitness on the training and validation set, its size and the total computation effort. The computation effort is measured as the total number of equation evaluations performed in both fitness calculations. In all figures and discussions, the exact fitness reported is computed on the validation data set. The evolutionary algorithm was stopped if convergence was detected; otherwise, the algorithm was stopped if it reached 10^{10} evaluations, which is equivalent to approximately 2.5 million generations.

V. Experimental Results and Discussion

First, the fitness of the best solution is measured with respect to the computational effort for each trainer strategy (Fig. 4). All the strategies have the similar trends, suggesting that each solution population evolves in a comparable manner. Due to the quantization of trainer updates, the fitness of the best solution does not increase until after the first update in

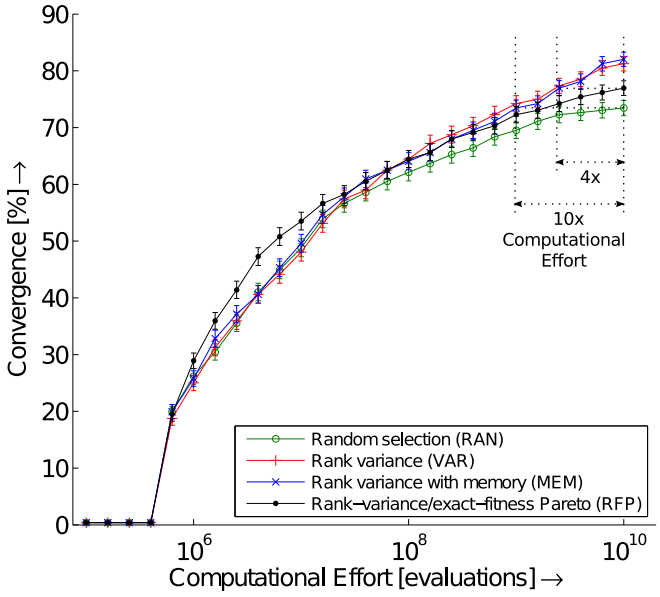


Fig. 5. Convergence rate of the trainer selection strategies averaged across the test suite versus computational error of the coevolution algorithm. The error bars indicate standard error of the mean.

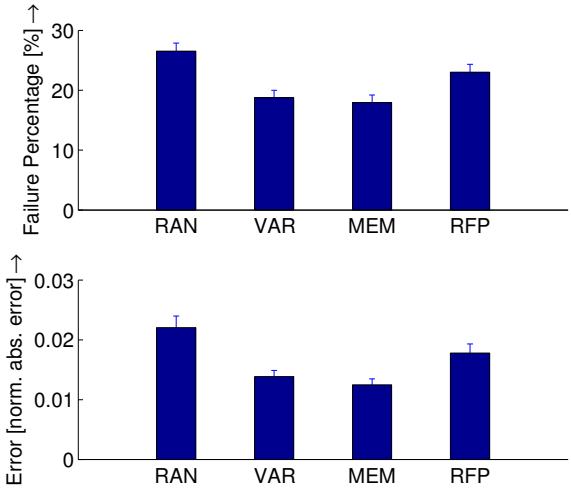


Fig. 6. The failure percentage and solution error for the strategies at 10^{10} evaluations. The error bars indicate standard error of the mean.

the trainer population, which occurs around $10^{5.8}$ evaluations. After the initial trainer update, the quantization effects become less significant and the error decreases in a logarithmic fashion.

Although the trends are similar, there are important differences in the overall performance of the trainer strategies. As a baseline, *RAN* achieves the worst performance. The two strategies, *VAR* and *MEM*, behave in nearly identical fashions and achieves the best results by 10^{10} evaluations. The *RFP* strategy outperforms all the other strategies at the beginning of the experiment, but ends up inferior to the *VAR* and *MEM* strategies as the computational effort increases.

A complementary trend is also observed when comparing convergence to computational effort between the trainer selection strategies (Fig. 5). The *RAN* strategy performs the worst, while the *RFP* strategy takes an early lead but is ultimately inferior to the *VAR* and *MEM* strategies. In fact, the *VAR* and *MEM* strategies are able to achieve the same final convergence percentage four to ten times faster than the *RFP* and *RAN* fitnesses, respectively. Also, it should be noted that every trainer selection strategy converged on the true solution orders of magnitude faster than the solitary evolutionary algorithm without predictor coevolution and is not shown.

The final performance of the strategies, with respect to the convergence and fitness metrics, are summarized in Fig. 6. The difference in performance is clear – the *VAR* and *MEM* strategies achieves a 1.5 times better failure rate than the baseline *RAN* strategy, while the *RFP* strategy fails 1.15 times less frequently. Similarly, the *VAR* and *MEM* strategies obtain 1.75 times less error than the *RAN* strategy, while the *RFP* strategy provides a a 1.25 fold decrease in error.

These results are surprising and indicate that there is unexpected, emergent behaviour from the tightly coupled populations. First, by comparing the *RAN* strategy with the remaining strategies, it is clear that intelligent subsampling does perform significantly better than random selection as the *RAN* strategy consistently achieves the worst results.

Next, the *VAR* and *MEM* strategies obtains the best performance by the end of the experiment in both fitness and convergence metrics. This suggests that selecting for solutions based on rank variance is a meaningful strategy as it allows the predictors to fully exploit the competitive dynamics in the coevolutionary algorithm.

Comparing the *VAR* and *MEM* strategies with each other suggest that memory does not play an important role in performance. The results imply that updating the entire trainer population in a single step or in a serial fashion does not have a significant effect on the emergent behaviour of the system. As a consequence, the *MEM* strategy is suggested as it only requires a single search for the solution of highest rank variance, while the *VAR* strategy requires multiple searches to update and fill the entire trainer population, resulting in a slightly larger overhead that is not reflected computational metric of fitness evaluations.

However, there are two parameters which could create discernible differences between the *VAR* and *MEM* strategies. The first parameter is the total run time – the performance of the two strategies might diverge if the termination condition was extended beyond 10^{10} evaluations. The second parameter is the trainer update period – this parameter was set to 100 generations for convenience and further characterization with respect to this parameter was not conducted. Changing this parameter might tease out some intricacies in the emergent, coevolutionary behaviour and is left for future work.

Finally, the *RFP* strategy provides an interesting point of analysis when compared to the *VAR* and *MEM* strategies. The *RFP* is able to initially achieve superior performance but is unable to maintain its advantage as the computation effort increased. These results suggest that the *RFP* is more likely

to convergence prematurely and get trapped in local optima. By adding selective pressures to model high exact fitnesses, the coevolutionary algorithm is too occupied with exploitation and does not conduct sufficient exploration.

Conversely, the results seems to suggest that elitism on exact fitnesses is not essential, but can be actually relatively detrimental, for coevolutionary progress. The *VAR* and *MEM* strategies do not encourage any elitist behaviour in their predictors, and as a consequence, this may allow the predictor and solution populations to bootstrap each other out of local optima. However, using exact fitnesses may not be the best method to promote elitism in the trainer population. A strategy to combine the early exploitation of *RFP* and the long-term exploration of *VAR*/*MEM* is left for future work.

VI. Conclusion

The goal of this work was to investigate different trainer selection strategies and their effects on coevolved rank predictor systems. Previous work on predictor coevolution focused on improving the predictor performance, often leaving the trainer selection strategy as an afterthought. Four different strategies were presented, ranging from random subsampling to selecting rank-variance/exact-fitness Pareto optimal solutions.

The four trainer selection strategies were compared on a symbolic regression problem using hundreds of randomly generated test problems, varying in problem complexity and number of variables. Of the four strategies, the *highest variance* strategies, with and without memory, was shown to provide superior results. It was able to achieve higher fitnesses and better convergence, identifying the true model four to ten times faster than the alternative strategies.

Acknowledgement

This work was supported in part by NIH NIDA grant RC2 DA028981, NSF CDI Grant ECCS 0941561, and DTRA grant HDTRA 1-09-1-0013. D. L. Ly would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for their support through the PGS program. The content of this paper is solely the responsibility of the authors and does not necessarily represent the official views of the sponsoring organizations. We also acknowledge Michael D. Schmidt for his insightful discussions and feedback.

References

[1] R. Poli, W. B. Langdon, and N. F. McPhee, *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008.

[2] Y. Jin, "A Comprehensive Survey of Fitness Approximation in Evolutionary Computation," *Soft Computing Journal*, vol. 9, pp. 3–12, 2005.

[3] Y. S. Ong, P. B. Nair, and A. J. Keane, "Evolutionary optimization of computationally expensive problems via surrogate modelling," *AIAA Journal*, vol. 41, pp. 687–696, 2003.

[4] D. Song, M. I. Heywood, and A. N. Zincir-Heywood, "Training Genetic Programming on Half a Million Patterns: An Example from Anomaly Detection," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 225–240, 2005.

[5] M. D. Schmidt and H. Lipson, "Coevolution of Fitness Predictors," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 6, pp. 736–749, 2008.

[6] ——, "Predicting Solution Rank to Improve Performance," *Genetic Evolutionary Computation Conference (GECCO)*, pp. 949–956, 2010.

[7] B. Johanson and R. Poli, "GP-music: an interactive genetic programming system for music generation with automated fitness raters," *Conference on Genetic Programming*, pp. 181–186, 1998.

[8] J. Branke, C. Schmidt, and H. Schmeck, "Efficient fitness estimation in noisy environment," *Proceedings of Genetic and Evolutionary Computation*, pp. 243–250, 2001.

[9] K. H. Liang, X. Yao, and C. Newton, "Evolutionary search of approximated n-dimensional landscape," *International Journal of Knowledge-based Intelligent Engineering Systems*, vol. 4, no. 3, pp. 172–183, 2000.

[10] R. E. Smith, B. A. Dike, and S. A. Stegmann, "Fitness inheritance in genetic algorithms," *ACM Symposium on Applied Computing*, pp. 345–350, 1995.

[11] M. Pelikan and K. Sastry, "Fitness inheritance in the Bayesian optimization algorithm," *Genetic Evolutionary Computation Conference (GECCO)*, pp. 48–59, 2004.

[12] L. T. Bui, A. A. Hussein, and D. Essam, "Fitness inheritance for noisy evolutionary multi-objective optimization," *Genetic Evolutionary Computation Conference (GECCO)*, pp. 779–785, 2005.

[13] H. S. Kim and S. B. Cho, "An efficient genetic algorithm with less fitness evaluation by clustering," *IEEE Congress on Evolutionary Computation*, pp. 887–895, 2001.

[14] Y. Jin and B. Sendhoff, "Reducing fitness evaluations using clustering techniques and neural network ensembles," *Genetic Evolutionary Computation Conference (GECCO)*, pp. 688–699, 2004.

[15] A. Teller and D. Andre, "Automatically Choosing the Number of Fitness Cases: The Rational Allocation of Trials," *Conference on Genetic Programming*, pp. 321–328, 1997.

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

[17] L. A. Albert and D. E. Golberg, "Efficient Discretization Scheduling in Multiple Dimensions," *Genetic Evolutionary Computation Conference (GECCO)*, pp. 271–278, 2002.

[18] C. D. Rosin and R. K. Belew, "New Methods for Competitive Coevolution," *Evolutionary Computation*, vol. 5, no. 1, pp. 1–29, 1997.

[19] B. Li, T. S. Lin, L. Liao, and C. Fan, "Genetic algorithm based on multipopulation competitive coevolution," *IEEE Congress on Evolutionary Computation*, pp. 225–228, 2008.

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

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

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

[23] M. D. Schmidt and H. Lipson, "Distilling Free-Form Natural Laws from Experimental Data," *Science*, vol. 324, no. 5923, pp. 81–85, 2009.

[24] J. Bongard and H. Lipson, "Automated reverse engineering of nonlinear dynamical systems," *Proceedings of the National Academy of Science*, vol. 104, no. 24, pp. 9943–9948, 2007.

[25] B. McKay, M. J. Willis, and G. W. Barton, "Using a tree structured genetic algorithm to perform symbolic regression," *Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp. 487–492, 1995.

[26] D. Edwin and B. P. Jordan, "Multi-objective methods for tree size control," *Genetic Programming and Evolvable Machines*, vol. 4, pp. 221–233, 2003.

[27] M. D. Schmidt and H. Lipson, "Comparison of tree and graph encodings as a function of problem complexity," *Genetic Evolutionary Computation Conference (GECCO)*, pp. 1674–1679, 2007.

[28] 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," *Australian Journal of Intelligent Information Processing Systems*, vol. 7, pp. 114–121, 2001.

[29] S. W. Mahfoud, *Niching methods for genetic algorithms*. Doctoral dissertation. University of Illinois at Urbana-Champaign, IL, 1995.