

Automatically Choosing the Number of Fitness Cases: The Rational Allocation of Trials

Astro Teller*

Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
(412) 268-7123

David Andre†

Division of Computer Science
University of California at Berkeley
Berkeley, CA 94720
(510)-643-6228

ABSTRACT

For many problems to which genetic programming has been applied, choosing the number of fitness cases with which to evaluate the individuals is a crucial decision. If too few fitness cases are used, overfitting may occur, and the measured fitness of an individual may not be representative of its true fitness. On the other hand, if too many fitness cases are used, a great deal of computer time can be wasted. This paper presents a method for the Rational Allocation of Trials (RAT) that dynamically allocates a boundedly optimal number of fitness cases for each individual. RAT allocates individuals to tournaments prior to their evaluation, and then, borrowing from previous work in model selection, allocates trials (fitness cases) only to those individuals for whom the cost of evaluating another fitness case is outweighed by the expected utility that the new information will provide. For most evolutionary computation approaches, including genetic programming, and for most problems, the RAT algorithm will provide significant time savings at minimal additional system complexity.

1 Introduction

Many problems in genetic programming have the characteristic that many possible input sets exist, and performance on the problem will be measured by the ability of the evolved program to perform well on a general, possibly infinite set of fitness cases. It is often impractical or impossible to test an evolving program against all the possible situations in which it will have to perform. Thus, a small in-sample group of fitness cases is often

*This work is supported in part through an ONR Muri grant and Astro is supported through the generosity of the Fannie and John Hertz Foundation.

†David is supported through the generosity of a National Defense Science and Engineering Grant from the Department of Defense.

used for the evaluation of fitness during evolution – with the hope that the evolved solutions will generalize. However, the question of how many in-sample fitness cases to use to obtain good generalization at low cost is difficult, because no theoretical results exist for most of the problems to which GP is applied. If too few fitness cases are used, the measured fitnesses of individuals may not reflect real fitness values, and the ranking and selection of individuals in the evolutionary process will be incorrect.

In general, most evolutionary approaches take a conservative approach to avoid this problem, and choose a number of fitness cases that is large enough that the measured in-sample fitness reflects out of sample performance. However, this wastes a great deal of computational resources. To avoid this problem, we suggest a method for the Rational Allocation of Trials (RAT) that adaptively selects the number of fitness cases for each individual in the population. RAT uses the simple mathematics of sample statistics to calculate the utility of evaluating another fitness cases for an individual, and is based on previous work in the machine learning literature on model selection [Moore and Lee, 1994]. Given some simple assumptions, the RAT algorithm results in a *boundedly optimal*¹ allocation of fitness cases for individuals. In addition, this paper presents the notion of operator utility models, which are crucial to the RAT algorithm and are potentially also a useful theoretical tool for understanding evolutionary computation.

In the RAT algorithm, a new generation is created by first (prior to the evaluation of fitness) filling all of the tournaments with M individuals, chosen randomly from the entire population. Then, some small number of fitness cases, T_{min} , is evaluated. After these initial cases, individuals are only evaluated on new fitness cases when there is some chance that they might win some tournament that they are losing or lose some tournament they are winning and if it is worth the cost of evaluation to prevent a loss of utility due to inaccurate measures of fitness. The key idea is that if an individual has no chance of winning a tournament, or if an individual is virtually guaranteed to win a tournament, no further fitness cases need to be evaluated.

¹Boundedly optimal because an upper-bound, as a function of the RAT parameters, can be determined on the error caused by RAT.

2 Related Work

Early stopping criteria is a well traveled subject in the statistics literature. In AI, and specifically, in machine learning, work with some of the same general goals or methods as this work include [Gratch *et al.*, 1993; Greiner and Jurisica, 1992; Moore and Lee, 1994]. Because this paper uses genetic programming as the example domain for RAT, we will give two examples of related work from that field.

[Maxwell, 1994] describes how, under certain conditions, it is possible to allow evolving programs to run for varying lengths of time and to give each a fitness score per execution time. This is an example of how people in the evolutionary computation community have, in the past, attempted to address issues of training time, though in this case the question is “How long should I let a program run on a particular training case” and not directly, “How many training cases should I give this program?”

[Tackett, 1994] describes a process called “Brood Selection” in which many children are created and then a pre-selection goes on in an attempt to weed out all but the most fit children of recombination. This is another way of addressing the issue of the number of training cases, but in this work there is no attempt at optimality and brood selection inherently uses a *separate* process of fitness attribution for the children in the weeding-out step of the selection process.

Chapter 5 of [Holland, 1992] is entitled “The Optimal Allocation of Trials” and while the theory is described in the context of the k-armed bandit problem, the next chapter shows its application to genetic algorithms. So, if there is already an optimal allocation of trials, why do we need a procedure for the rational allocation of trials?

3 Holland’s Optimal Trial Allocation

Holland showed the optimal schedule for exploring and exploiting among a variety of choices in which each choice produces a probabilistic pay-off. He described this general problem in terms of deciding which of k arms to pull on a k -armed slot machine (bandit) whose arms are not all equally likely to pay-off (i.e. spit out coins). The crucial assumption in Holland’s analysis is that when you get the pay-off, you are richer by exactly that pay-off! In other words, when the k-armed bandit spits out c coins on pull i , you count them, keep them, and even if pulling some other arm on that turn would have paid off better, it will never turn out later that you, in retrospect, got either fewer or more than c coins on turn i .

Holland’s optimal schedule tells you how much to bias your search towards the candidates that look better right now. This optimal trade off between exploration and exploitation is possible because the cost of exploring can be approximated – it is the lost payoff from not choosing the highest paying arm! However, if the payoff of an arm (model) cannot be known until long after the decision about the model must be made, the cost associated with choosing a model cannot be estimated in the same way.

The aim of our work is to address this issue. In a run of evolutionary computation, if we find out the value of a particular model from an intermediate generation, it will only be at the end of the run, when the final solution emerges. As is described in section 7, the utility of a particular piece of information in the evolutionary context is a complex issue.

4 The RACE Algorithm

The RACE algorithm is described in [Moore and Lee, 1994]. The goal of the RACE algorithm is to save time in the process of model selection. Given a large number of models to choose from and a large number of training examples to train each model, it would be nice to have a principled way to stop training models as it becomes clear that they will not turn out to be the single best model to choose (i.e., the one with the lowest error).

The idea is to approximate with a Gaussian² the mean and variance of the error produced by each model. Once we know these values, we can discard any model if the probability is pretty high that there is another model whose mean error is at most only slightly larger. We will call these thresholds γ (slightly larger) and $(1 - \delta)$ (pretty high). Here is an outline of the RACE algorithm that, given T training examples, picks one of M models that has the lowest error:

1. Pick an error tolerance, γ , and a probability tolerance, δ , such that we will stop examining M_i if there exists another model M_j such that the chance is at least $(1-\delta)$ that e_i^* is no lower than $(e_j^* - \gamma)$. (Where e_i^* is the real error for model M_i .)
2. Make the *strong* assumption that the errors a model makes on each trial are normally distributed.
3. As k goes from 1 to T :

- (a) Error **Mean** for model j :

$$\hat{\mu}_{kj} = \frac{1}{k} \sum_{i=1}^k e_j(i) \quad (1)$$

- (b) Error **Standard Deviation** for model j :

$$\hat{\sigma}_{kj} = \sqrt{\frac{1}{k-1} \sum_{i=1}^k (e_j(i) - \hat{\mu}_{kj})^2} \quad (2)$$

- (c) Drop any model M_i on this iteration (k) such that:

$$\text{Prob}(e_i^* < e_j^* - \gamma) < \delta \quad (3)$$

for some model M_j on this same iteration (k).

In general, if we have independent random samples, of sizes k_i and k_j from different distributions with sample

²For non-ordinal (e.g., binary) model error values, other distributions (e.g., weighted binomial) should be used.

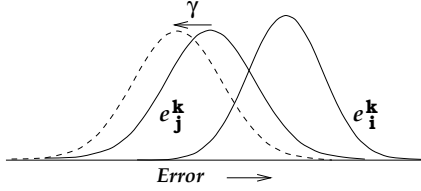


Figure 1: Visualization of equation 3: e_j^k and e_i^k are the error distributions for iteration k that estimate the mean errors ($\hat{\mu}_{kj}, \hat{\mu}_{ki}$) and the confidences in those means ($\hat{\sigma}_{kj}, \hat{\sigma}_{ki}$). Equation 3 shifts e_j^k down the error scale by exactly γ .

means μ_i and μ_j and standard deviations σ_i and σ_j , then we can re-express equation 3 as³:

$$\text{Prob}(x < 0) < \delta \quad (4)$$

$$\mu_x = \mu_j - \mu_i - \gamma \quad (5)$$

$$\sigma_x^2 = \frac{\sigma_j^2}{k_j} + \frac{\sigma_i^2}{k_i} \quad (6)$$

x is a normally distributed variable with mean μ_x and standard deviations σ_x . Equation 4 can be computed as a simple lookup table. For the RAT algorithm, $k_i = k_j$.

This strategy will throw out all but the best model (within an error tolerance of γ) with probability at least $(1 - \delta)^M$ (where M is the number of models). This strategy will also throw out very similar models (those whose real error difference is within the error tolerance γ) as long as their variance is small relative to their error difference. However, when two models have similar error means and large error variances, they may race indefinitely under this strategy, particularly in the extreme case in which they are identical.

The goal of the RACE algorithm as Moore and Lee describe it is to discard all but the best model as quickly as possible. Keep in mind that in RAT we will not be “throwing out” a population individual, but only ceasing to improve our approximation to its actual fitness. That individual might easily be highly represented in the next generation.

5 The BRACE Algorithm

The BRACE algorithm, described in [Moore and Lee, 1994], deals with the problem of RACEing similar models with high variance by appealing to the Blocking statistical technique [Box *et al.*, 1978]. The Blocking insight is that, by confining sample comparisons to within “blocks” (groups of more homogeneous data), greater precision can often be obtained. Equations 7, 8, and 9 are the relevant Blocking formalisms for BRACE. BRACE (blocking race) is, simply put, the RACE algorithm in which the Blocking algorithm is used as the criteria for stopping (rather than equation 3).

Let us define a value of response similarity, h_{ij}^* :

$$h_{ij}^* = e_i^* - e_j^* \quad (7)$$

³subject to an approximation that is valid for $k_i, k_j \geq 20$.

Then we can measure the mean and variance of this value as it is sampled at each training instance k . We will define $\hat{\mu}_{ij}^h(k)$ and $\hat{\sigma}_{ij}^h(k)$ as:

$$\hat{\mu}_{ij}^h(k) = \frac{1}{k} \sum_{l=1}^k (e_i(l) - e_j(l)) \quad (8)$$

$$\hat{\sigma}_{ij}^h(k) = \sqrt{\frac{1}{k-1} \sum_{l=1}^k ((e_i(l) - e_j(l)) - \hat{\mu}_{ij}^h(k))^2} \quad (9)$$

Now, as k goes from 1 to T , we can eliminate a model M_i on iteration k when:

$$\text{Prob}(h_{ij}^* < -\gamma) < \delta \quad (10)$$

for some model M_j on this same iteration k .

[Moore and Lee, 1994] present these two algorithms as methods for quickly choosing a single best model. Evolutionary computation has a similar problem to that of model selection – picking multiple *better* individuals from a population. Evolutionary computation often has more data than is needed to provide an adequate sorting of the population and it would be nice to have a principled way to allocate trials in order to minimize the number of trials needed for an accurate sort. This is particularly important because the calculation of fitness is the most computationally expensive part of each generation. However, several aspects of the evolutionary process require a more complex decision criteria for evaluating the relative quality of individuals in an evolving population.

6 RAT

There are three major complications in using BRACE for the selection of individuals in evolutionary computation for the task of efficient trial allocation. The first is that, for some reproduction strategies (e.g., “roulette wheel”) knowing that individual X is more fit than individual Y with high probability is not enough since the strategy operates on the *amount* of fitness that separates X and Y . The second complication is that the evolutionary process, by definition, picks not one, but many “models” (i.e. population individuals).

The third, and most serious complication in applying BRACE to evolutionary computation, is the choosing of γ (the fitness tolerance used to balance the cost of an additional trial). For normal model selection, the fitness of a model simply represents the model’s expected utility. For evolutionary computation, on the other hand, the expected utility of an individual is not necessarily based on its fitness – its utility is its usefulness in further search for a problem’s solution. An individual’s expected utility may be related to its fitness, but is also potentially related to other factors, such as genetic diversity. Thus, in choosing between two individuals X and Y whose fitness difference is $F_X - F_Y$, we should choose on the basis of an error tolerance γ that is dependent in some way on F_X and F_Y and perhaps other factors. For now, we describe the RAT algorithm using a γ as before, and in section 7.2 we will fully address various γ functions.

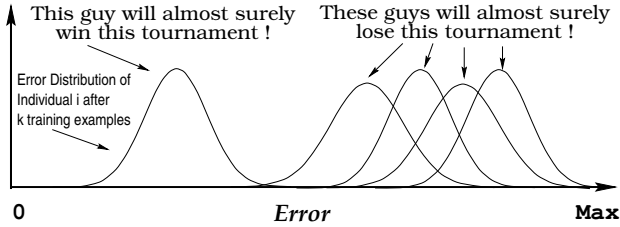


Figure 2: An example set of error/fitness distributions for a tournament of size $S = 5$.

We will first respond to the first two problems that can both be addressed with the same mechanism. The “amount of fitness” problem can be avoided by using a reproduction strategy that does not have this as a parameter. Examples of this type are *rank selection* and *tournament selection*. Tournament selection is probably the most popular form of reproduction in evolutionary computation. Largely for that reason, RAT will be explained in the context of tournament selection. However, RAT could be accomplished using any selection strategy in which the “exact fitness amount” factor has been removed. Tournament selection also solves the second problem, picking many models instead of one, by subdividing the selection problem into many small selection problems in each of which only one model will be selected.

In the RAT algorithm, a new generation is begun by first (prior to the evaluation of fitness) creating M_{mp} tournaments, each of which is filled with S randomly chosen individuals from the current population. Then, some small number of fitness cases, T_{min} , is evaluated. After these initial cases, individuals are only evaluated on new fitness cases when there is some chance that they might win some tournament that they are losing or lose some tournament they are winning (see figure 2) and if it is worth the cost of evaluation to prevent a loss of utility due to inaccurate measures of fitness. The key idea is that if there is little benefit to further refining the approximation to an individual’s fitness (i.e. if its fate is essentially sealed) then no further fitness cases need to be evaluated.

The general algorithm for tournament selection is, given a population of size M , a mating pool of size M_{mp} , and a “tournament size” of S :

1. Do M_{mp} times
 - (a) Pick S individuals at random from Population;
 - (b) From this set, place a copy of the individual with highest *approximated* fitness into the “mating pool”
2. Population $_{i+1}$ is created by the application of genetic operators to the mating pool

The basic RAT algorithm works as follows:

1. Pick M_{mp} tournaments, *before generation begins*.
2. Initialize contention list (Q) with all individuals
3. Do T_{min} times

- (a) evaluate all individuals on current example (fitness case)

4. Do $T - T_{min}$ times

- (a) remove any individual X from Q if:
 - for every tournament t that X is in:
 - X is not in first place AND

$$\text{Prob}(h_{X P_{tx}^{min}}^* < -\gamma) < \delta \quad (11)$$

OR

X is in first place AND

$$\text{Prob}(h_{P_{tx}^{min} X}^* < -\gamma) < \delta \quad (12)$$

- (b) evaluate all individuals still in list Q on the current training example

In the algorithm above, P_{tx}^{min} is, for a given tournament t and individual X , the distribution on the minimum of the fitnesses of the individuals other than X in tournament t . Notice that we do not update the mean and variance on the fitness distributions for each population member as the fitness cases are evaluated. This would be the RAT version of RACE; RAT is, instead, an extension of BRACE.

There are three important points to be made about this version of the RAT algorithm. First, understand that when an individual X in the population is removed from Q , this is **not** an indication either that the individual is bad or that it will not positively contribute to the evolutionary process. It only means that RAT judged there to be too little value in further information *about* that individual to justify further fitness approximation on that individual before the reproduction process begins.

Second, the time it takes to decide whether, on iteration k of generation G , individual X should be removed from Q is $O(M'S)$, where M' is the number of tournaments that X is in. M' has an upper bound of M and an expected value of slightly less than $\frac{M_{mp}}{M}S$. Since $\frac{M_{mp}}{M}$ is never larger than 2, this means that the major time cost for RAT is approximately quadratic in the tournament size. Since the conventional wisdom sets S to $\log M$ or $\log \log M$, the additional computational expense incurred by RAT is very sub-linear in the size of the population. This is a small price to pay relative to the saving of dropping out individuals once their reproductive fate has been determined.

Third, this version of RAT assumes that each fitness case contributes equally to a composite fitness value. There are, of course, domains in which some fitness cases are “more important” than others. If you know *how much* more important these cases are, RAT can be trivially adjusted to include a multiplier on each fitness case. If you know that some fitness cases are “more important”, but you don’t know which ones those are, then RAT as just described is an acceptable approximation to the extent that the uniform distribution of fitness importance is an accurate one.

7 Utility Optimality in RAT

This section presents the difficulties of measuring the utility of individuals in evolutionary computation and presents a framework for making rational decisions about fitness trial allocation at several different granularities of assumptions about expected utility. In this section we will refer to the situation of choosing between two individuals X and Y whose fitness difference is $F_X - F_Y$ using RAT with various functions for (γ) . Keep in mind that X and Y are never directly compared by RAT. Each individual is compared only against the rest of its tournament. This section focuses on the relative utility of two possible outcomes: X wins tournament t or Y wins tournament t .

7.1 The Difficulty of Measuring Utility in Evolutionary Computation

In the field of evolutionary computation, the goal of a run of simulated evolution is most often to produce a single individual that represents a solution to the problem at hand. Given this, all payoff is at the end of the evolutionary process of many generations – the “utility” of a specific characteristic of the system must be measured in terms of the likelihood that the characteristic improves the quality of the solution. In comparison, for techniques similar to RAT that have been highly studied, such as single model selection and the multi-arm bandit problems, determining the utility of an individual is relatively straightforward. For the case of one-time selection of a single model, the measured fitness of the individual (model) is the individual’s utility, as the individual can be utilized immediately after selection. For the multi-arm bandit problems, reward is doled out over many iterations, and is summed cumulatively. When rewards occur, the cause of the reward is obvious, and the statistics for the appropriate “arm” can be adjusted. In techniques like evolutionary computation, however, the problem of model selection is embedded within the process of iterated model selection, and the utility of an individual in the middle of the process cannot be calculated exactly.

The usefulness of an individual is the likelihood that the individual, through its offspring (and their offspring and so on), will affect⁴ the eventual solution to the problem. This usefulness is not necessarily simple to calculate, and is dependent on the operations that will be performed on a given individual. For example, the utility of a given individual for mutation may be different than its utility for crossover. Let us call the tree of ancestors of the solution the *solution family tree* and say that an individual that directly affects the eventual solution is a part of the solution family tree. Although it may seem at first that every winner in a reproduction tournament has the same probability of affecting the output, utility

⁴An individual can *affect* the eventual solution individual either by having the solution individual as a descendent *or* by affecting the search process which in turn has an affect on *how* the solution individual is discovered.

is not so easy to calculate. For each individual in the population that wins a tournament, it can contribute to the next generation in one of three ways:

1. Copy: it moves over unchanged
2. Mutate: copy with intra-individual replacements
3. Crossover: copy with inter-individual exchanges

In the case of 1, a low fitness individual is much less likely than a high fitness individual to survive *the following generation* and so whether X or Y was picked is a less important question if X and Y were both low fitness individuals. In other words, the likelihood that a copy of a low fitness individual will be a part of the solution family tree is small, and thus the expected payoff is low.

In the case of 2 and 3, exploration takes place – the offspring is different from the parent – and it is very difficult to say whether X or Y has a greater likelihood of producing an offspring in the solution family tree. This determination is difficult because the expected fitness of the offspring of an individual created via mutation or crossover is a function of the behavior of the given operator, which in turn is a function of a great number of variables. For example, for two low fitness individuals, the main factor determining the utility of the individuals may be, rather than their fitness values, how behaviorally or syntactically distinct they are from the rest of the population.

7.2 The RAT Framework For Choosing γ Functions

We propose a series of γ functions with increasing complexity and with finer and finer granularity of assumptions about utility to the final solution quality. To the extent that the assumptions in each case are true for a given problem, then that γ function is rational. Now, to fully define a RAT system, one could choose γ functions for each of the search operations – i.e.: $\gamma_{\text{crossover}}$, γ_{mutation} , and γ_{copy} .

No γ ($\gamma(\dots) = 0$)

This γ function assumes that the cost of trials are insignificant and any difference in fitness is a sufficient condition for a RACE. In this case, the choice of whether to perform further fitness trials for an individual is based solely on the probability that h_{XY}^k is greater than zero.

γ is constant ($\gamma(\dots) = c$)

This γ function assumes that the utility of tournament winners is constant, and is similar to the γ function used by (B)RACE. In this case, γ should be chosen to balance the computational cost of another fitness trial, as well as the acceptable error/fitness tolerance.

γ is related to its expected “impact” ($\gamma(?_X, ?_Y)$)

In this case, γ is based on the expected number of copies of individual X in the mating pool. The idea is that for individuals with high fitness, the differences in fitness matter more (have a higher probability of affecting the eventual solution) than do the differences for lower fitness individuals. This difference is assumed to be related

to the expected number of tournaments that an individual should win based on its fitness rank. The expected number of copies of individual X in the mating pool, if its fitness rank (M is best) in the population is R_X , is:

$$k\left(\frac{R_X}{M}\right)^{k-1} \quad (13)$$

With equation 13, we can sophisticate the γ function:

$$\gamma(R_X, R_Y) = \frac{\gamma}{1 + (k\left(\frac{R_X}{M}\right)^{k-1} + k\left(\frac{R_Y}{M}\right)^{k-1})/2} \quad (14)$$

We could also actually count the number of tournaments that X and Y are in (T_X and T_Y respectively). Then equation 14 becomes:

$$\gamma(T_X, T_Y) = \frac{\gamma}{1 + (T_X + T_Y)/2} \quad (15)$$

T_X and T_Y do not reflect how many tournaments X and Y will respectively win. RAT can progressively take this additional information into account in T_X^k and T_Y^k as follows:

1. When $k = 0$, $T_X^k = T_X$
2. When X is removed from Q , $T_X^k = T_X - L$, where L is the number of tournaments X is predicted to lose.

γ functions are derived from operator behavior on previous problems

In this case, we will be examining utility at a finer level of detail. Rather than assuming an individual's utility to the final solution quality is constant or a simple relation to the number of tournaments that that individual is in, this case assumes that an individual's utility is a potentially complex function of its characteristics, the search operator type, its partner's characteristics if the search operator requires a partner, on the generation, and on the problem being solved. On previous runs of similar problems, the behavior of operators would be measured, and then operators models built. A function $utility(F_X, gen, \dots)$ that maps fitness into expected utility could then be used to build appropriate γ functions.

As an example, it may be the case that when a high fitness individual is chosen as the father for a crossover, and a low fitness individual is chosen as the mother, then it is very unlikely to produce offspring with high fitness. This pairing would have low expected utility, and thus γ could be larger. The operator models that would result from such an analysis could be very useful for a greater theoretical understanding of crossover.

γ is learned on the fly

In this case, the γ functions that will be used on a given generation are calculated from the behavior of the operators over the previous few generations. This models the utility at even a finer degree of granularity, as it assumes that the utility of a crossover with parents at fixed fitness percentiles might change over the course of a run. It might be the case that performing more exploration earlier in a run is beneficial, and thus γ should be higher,

perhaps, earlier in the run. In this case, the utility of the various operators would be measured (as in the previous case) and γ functions created for the current generation.

γ is a function of multiple factors

Finally, it may be the case that it is necessary to measure the utility of various operations for various individuals by examining variables other than fitness, number of tournaments (which is directly related to fitness), and generation number. As previously noted, it might be the case that for high fitness individuals, fitness is the best predictor of utility, but for low fitness individuals, other factors, such as diversity or parsimony, might be much more significant. This method for determining γ functions removes the assumption that γ be some function of only the fitness.

7.3 In General

In general, the appropriate settings for γ (and δ), and the choice of whether to make them constants or functions of the learning technique being used is an important one. In any learning paradigm in which model selection happens many times in series during the process, *how* the selected models are used in the remainder of the process should clearly effect what strategies are used to adjust the selection of each of those models. In this section, we have tried to convey a few of the options and complications that this issue takes on in the specific case of evolutionary computation.

8 Experimental Demonstration

RAT has been presented as a general method for dramatically and *correctly* (within the adjustable bounds) speeding up genetic programming (and evolutionary computation in general). While RAT stands on its own as a procedure, a practical example of its application will demonstrate RAT's usefulness and highlight another way of thinking about its benefits.

8.1 Experimental Setup

For the purpose of demonstration, we chose a simple domain familiar to the GP community and reasonable, fixed values for δ and γ . We selected symbolic regression as the test problem type and the Gudermannian function as the curve to fit.⁵ For δ we used 0.01, and for γ we used 0.001. The population size (M) was fixed at 100 and each run included in the results below ran for 25 generations.

Figures 3 and 4 show two different costs incurred with and without RAT, averaged over ten runs. The independent variable in both graphs is the training set size. For all ten runs, $T_{min} = 20$ (see section 6).

8.2 Results

The most important collective feature of figures 3 and 4 is that RAT saves an enormous amount of the expense of evolution even for very reasonably sized training set

⁵The Gudermannian is $(\log((1/\cos(X)) + \tan(X)))$.

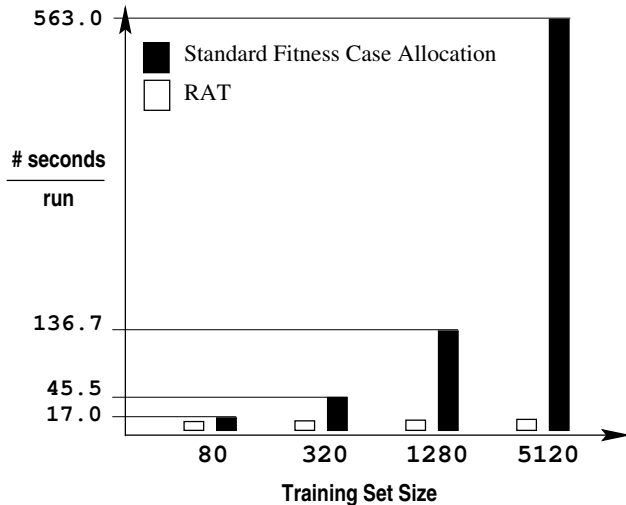


Figure 3: The average computational cost per run, with and without RAT

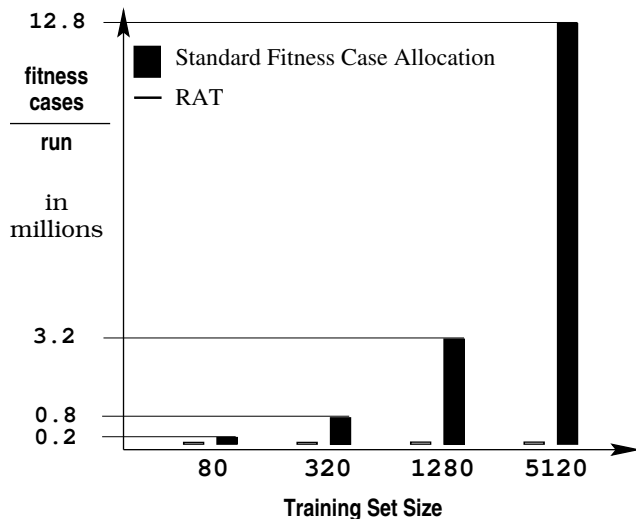


Figure 4: The average number of fitness cases evaluated per run, with and without RAT

sizes. An appropriate question is “RAT saves time at what cost to the evolved solution?” The good news is that the system performance (i.e. the quality of the evolved solution) with and without RAT were statistically indistinguishable.

Notice that because RAT incurs almost no additional computational cost after further training set items are unnecessary, RAT can be made to look arbitrarily preferable to a fixed training set strategy by simply increasing the training set size accordingly. So, rather than describe RAT as K times better than a fixed training set size policy, it is more accurate to think of RAT as helping to *pick a training set size* and to do this selection *per individual* instead of per run, as is traditional.

For this symbolic regression problem, RAT requires about 25 to 30 fitness cases for most individuals, no matter how many fitness cases are available. It is interesting

to note that dynamic allocation of fitness cases does occur. Some individuals were given upwards of 100 fitness cases, several times the average allocation. In addition, on some generations, the majority of individuals were allocated many fewer than 30 fitness cases, and in other generations, many more.

9 Discussion

In addition to these extra memory costs, RAT introduces some complexity for several commonly used optimizations in genetic programming. For example, because the memory costs of storing a population of thousands of programs are quite high, the individuals are often stored in a compressed format and then expanded when they need to be evaluated. Usually, each program is completely evaluated once it is expanded, so that at most one expanded program need be stored. However, the point of RAT is that individuals are not completely evaluated at once, so individuals must be uncompressed and compressed multiple times. Although compression and uncompression is relatively fast in comparison with the evaluation of an individual, there is some cost incurred. This problem can be almost entirely avoided by executing C fitness cases each time an individual is uncompressed. This amortizes the cost of the decompression and no individual will ever receive more than $C-1$ “extra” fitness cases.

The other GP technique affected by RAT is that of point typing (or other strong typing methods) where the choice of a point from one parent in crossover affects the choice of the second parent. In the presented algorithm for RAT, the tournaments for all parents are chosen at once. In order to do point typing, two “rounds” of RAT must be run – one for choosing the first parent of crossover and all of the single parent operations, and one for choosing the dependent parents.

There is a memory cost that must be paid for such an impressive reduction in time cost. RAT requires $O(F_c * M)$ memory where M is the population size and F_c is the average number of fitness cases an individual participates in during a generation. However, since RAT’s value lies in exactly that F_c is essentially constant no matter how large the training set size becomes, in practice this memory cost is negligible.

10 Current and Future Work

Although section 7.2 suggested many different models for how to handle the utility problem in RAT, it remains an open question as to which models will be sufficient to handle a wide variety of problems. Detailed empirical research on the effectiveness of these techniques is still underway. In section 8, we gave evidence that the RAT algorithm can, for reasonably sized training sets, speed up genetic programming by one or more orders of magnitude (figures 3,4).

In addition, the operator models suggested here are being used to examine the role of crossover in EC. Future work will also investigate the possibility that the rational

choice model can be used to determine factors in EC other than trial allocation, including population size and reproductive operator frequencies.

11 Conclusions

This paper presented RAT, a procedure for the Rational Allocation of Trials, that dynamically allocates a boundedly optimal number of fitness cases for each individual. RAT was explained in the context of and as an extension to the RACE and BRACE algorithms for model selection. In brief, RAT allocates individuals to reproduction tournaments prior to their fitness evaluation, and then allocates fitness trials to individuals only when the cost of evaluating another fitness case is outweighed by the expected utility that the new information will provide. Section 7 described some of the issues that this utility pressure introduces and proposed some methods of investigating these issues.

The RAT procedure is a speed up mechanism in evolutionary computation, not a qualitative improvement in the evolutionary computation itself. However, the number of fitness cases with which to evaluate population individuals is often a crucial decision: using too few fitness cases can cause overfitting; using too many fitness cases can waste significant computer time.

For most evolutionary computation approaches, including genetic programming, and for most problems, the RAT algorithm will provide significant time savings (a factor of 36 is shown in figure 3) at low extra additional programmatic complexity. The time savings will depend on the total number of training examples available but, for large available training sets and γ values near zero, RAT cheaply and automatically sets an EC-critical parameter to a near optimal value on a per individual basis.

References

- [Box *et al.*, 1978] G. E. Box, W. G. Hunter, and J. S. Hunter. *Statistics for Experimenters*. Wiley, 1978.
- [Gratch *et al.*, 1993] J. Gratch, S. Chien, and G. DeJong. Learning search control knowledge for deep space network scheduling. In *Proceedings of the 10th International Conference on Machine Learning*. Morgan Kaufman, 1993.
- [Greiner and Jurisica, 1992] R. Greiner and I. Jurisica. A statistical approach to solving the ebl utility problem. In *Proceedings of the 10th International Conference on Artificial Intelligence*. MIT Press, 1992.
- [Holland, 1992] J. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.
- [Maxwell, 1994] Sidney R. Maxwell. Experiments with a coroutine model for genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence, Orlando, Florida, USA*, volume 1, pages 413–417a, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.
- [Moore and Lee, 1994] A. Moore and M. Lee. Efficient algorithms for minimizing cross validation error. In *Proceedings of the 11th International Conference on Machine Learning*. Morgan Kaufmann, 1994.
- [Tackett, 1994] W. Tackett. The unique implications of brood selection for genetic programming. In *Proceedings of the First IEEE International Conference on Evolutionary Computation*, pages 160–3. IEEE Press, 1994.