

---

# Modeling Disk Arrays Using Genetic Programming

---

Evan Kirshenbaum

Hewlett-Packard Laboratories  
1501 Page Mill Road  
Palo Alto, CA 94304  
kirshenbaum@hpl.hp.com

## Abstract

This paper describes the results of using genetic programming to evolve models that predict the throughput in disk arrays. The results are compared to previous hand-crafted analytical and automatically-generated interpolation-based device models. An analysis is performed to investigate the optimality of the run parameters chosen as well as to discover whether the approach has the tendency to overfit its training data. The process is shown to find models that outperform both recently published and currently used models and to be sensitive to population size but not run length.

## 1 BACKGROUND

In the past decade, enterprises have turned to disk arrays both to give them sufficient storage capacity and also to help them attain their required availability levels. The internal architectures of the arrays—and those of the commodity disks they contain—are becoming more and more complex, now including hardware and firmware support for a wide variety of optimizations, typically developed independently of one another, leading to interactions that are often unintuitive and difficult to understand. (Uysal, et al., 2001)

To design and configure such systems, automated tools such as Minerva (Alvarez, et al., 2001) attempt to predict the performance of a large number (tens of millions) of configurations under various workloads to select an optimal design. Such tools require models which are accurate, execute extremely quickly, and are not inordinately expensive to build.

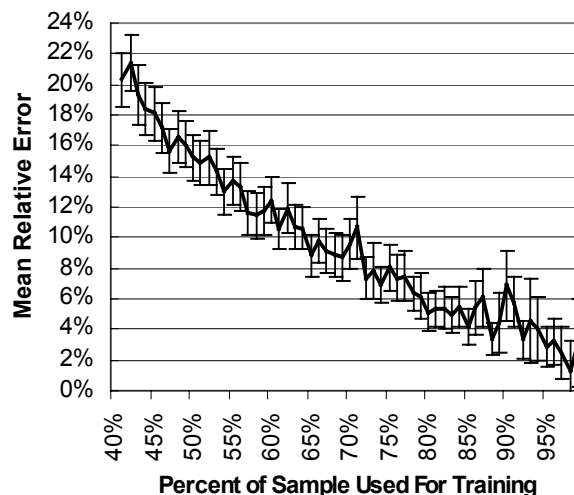
Uysal, et al. (2001) developed a modular analytical model for predicting the throughput of a disk array and validated it for a particular array under two configurations and a number of workload conditions, achieving an overall error rate of approximately 15%. Anderson (2001) investigated automatically constructing models based on table-driven interpolation between actual measured throughput values. He tried three algorithms: closest-point estimation,  $k$ -nearest-neighbor averaging, and hyperplane interpolation,

in which the prediction is based on interpolation over a non-degenerate hyperplane of nearby measured points.

Anderson empirically tested each of these methods on a data set obtained by measuring a disk array over two RAID levels, three numbers of disks, and a wide variety of workload configurations, for a total of 1,126 measurements. To simplify the task, he constructed a separate table for each combination of RAID level, number of disks, and operation type (read or write). Of the three methods, closest-point estimation was clearly the worst, and hyperplane interpolation appeared to be somewhat better than  $k$ -nearest-neighbor averaging.

Anderson's results for hyperplane interpolation for the 695 RAID level 1 cases can be seen in Figure 1 (a recasting of his figure 1), showing the mean relative error when constructing the tables using a given fraction of the available data and testing on the remainder, with fifty runs at each point. With 41% of the data, hyperplane interpolation yields a mean relative error of  $20.32 \pm 1.76\%$ . Performance steadily improves as the number of data points increases, reaching  $12.45 \pm 1.53\%$  with 60% of the data,  $5.11 \pm 1.22\%$  with 80% of the data and  $1.24 \pm 2.04\%$  with 98% of the data.

A downside to this approach, however, is the time it takes to acquire the data. Currently, each data point requires approximately five minutes of computer time to acquire,



**Table 1** Subset sizes

Subset	Cases
2-disk reads	72
4-disk reads	144
6-disk reads	144
2-disk writes	92
4-disk writes	99
6-disk writes	144

so the 556 cases used in the 80% test took over 46 hours to acquire. It is desirable to look for techniques which can achieve similar results while requiring less data. Genetic programming is a promising candidate, as it appears to be good at finding general solutions to non-linear functional numeric regression problems from a relatively small number of cases.

In this paper, we will describe the use of genetic programming (Koza, 1990, 1992) to solve the same problem. In section 2 we describe our experimental setup. In section 3 we present our results, paying special attention to the question of overfitting. In section 4 we look a bit closer at which of the experimental parameters seem to matter. We then finish up with some thoughts on future work and conclusions.

## 2 EXPERIMENTAL SETUP

The experiments described in this paper were performed on the RAID-1 subset of the data used in (Anderson, 2001). The data was partitioned into six subsets based on

**Table 3** Available Operators

Operator	Type
$+$ , $-$ , $*$ , $/$ $\min(x, y)$ , $\max(x, y)$ , $\text{mean}(x, y)$	Real x Real $\rightarrow$ Real
Constants $[-10, 10]$	Integer
Constants $[-10, 10]$ (step 0.0001)	Real
<i>queue length</i> <i>request size</i> <i>run count</i>	Integer
cache segment size (64 KB) cache size (256 MB) disk size (18 GB) max controller bandwidth (84 MB/s) max controller throughput (11,338 I/O /s) rotation time ( $6 \times 10^{-3}$ s) read position time ( $6.37 \times 10^{-3}$ s) write position time ( $7.00 \times 10^{-2}$ s) mean read transfer rate ( $1.80 \times 10^7$ B/s) mean write transfer rate ( $1.65 \times 10^7$ B/s)	Integer

**Table 2** Input data

Attribute	Description	Range
request size	The mean length of a request	2 KB to 256 KB
queue length	The mean size of the device queue	16 or 64
run count	The number of requests made to contiguous addresses	1 to 256

the number of disks (2, 4, or 6) and the operation being performed (read or write). The sizes of the subsets are shown in Table 1. For each data point, there are three input values available to the predictor, described in Table 2. In this data set, all input values were round powers of two. Each subset was separately trained on a randomly-chosen 20%, 40%, 60%, and 80% of the data, with the remainder used for testing, and ten runs were made at each combination, for a total of 240 runs.

The system used was GPLab, a flexible genetic programming framework developed and used for data mining research at Hewlett-Packard Laboratories. This system implements strongly-typed GP (Montana, 1995) with hierarchical types, so, for instance, operators (including variables and constants) which produce integers can be fed into those which require real numbers. Experiments were performed on a PC with a 1.3 GHz Pentium 4 processor.

The operators available to each run are shown in Table 3. In addition to the normal arithmetic operators over real numbers (with division by zero defined to return one), we also provided operators to take the minimum, maximum, and mean of two real numbers. Constants were provided, uniformly selected over the range  $-10$  to  $10$  inclusive, both as integers and real numbers. The run parameters themselves were available as integers. Finally, a set of “magic number” constants used by the analytic model in (Uysal, 2000) were included. Anderson (personal communication) informed us that the numbers we used may not actually describe the system on which the data was collected. In particular, the value for disk size is a bit too high and the cache segment size is a factor of four too large, with several of the others difficult to measure. Our expectation, however, is that the values, inaccurate as they might be, will be useful as “ballpark figures” providing seeds from which the actual values can be derived by combining arithmetically with numeric constants. The constants specific to reads and writes were only provided for runs that measured those operations.

Each run of the experiment lasted 300 reproductive generations with a population size of 5,000 candidates

Table 4 Results

Fraction	Mean	Median	Min	Max	Weighted Mean	Hyperplane Interpolation Mean
20%	$2.4 \times 10^9\%$	16.259%	8.233%	$4.0 \times 10^8\%$	$4.9 \times 10^8\%$	not given
40%	12.604±1.615%	11.401%	5.204%	52.925%	12.380%	20.317±1.764%
60%	10.164±0.881%	9.696%	3.396%	21.474%	10.066%	12.466±1.529%
80%	10.765±1.666%	9.788%	3.400%	28.019%	10.375%	5.106±1.221%

initially generated with a depth of no more than seven. In subsequent generations, (on average) 80% of the candidates were produced by crossover, 8% by copying, 8% by point mutation (replacing an operator in the tree by another operator with a compatible type signature), and 4% by constant drift (incrementing or decrementing an integer constant or applying Gaussian drift to a real constant). Traditional mutation was not used. The best candidate from a generation was unconditionally copied to the next. The primary fitness measure was the mean relative error over the training cases presented to the candidate during its training period. In case of exact ties, the number of nodes in the candidate's tree was considered as a secondary fitness measure. Candidates were selected to be parents by winning 5-candidate tournaments. There were no restrictions on the sizes of the trees resulting from crossover, but candidates who exceeded a 100-operator evaluation budget on any case were considered infinitely bad.

The runs made use of GPLab's *dynamic fitness case selection* feature both to avoid wasting time on very poor candidates and to minimize the risk of overfitting. Each candidate was presented with a set of ten cases drawn from the complete set of training cases. If it did sufficiently well on those cases, it was presented with another set of ten, and so on until it had seen the full set of training cases. "Sufficiently well" for this experiment was defined as scoring within a "hit interval" on 90% of the cases seen. The hit interval started at 15% and was reduced to be no more than 150% of the best score posted over all of the training cases by any candidate. A form of reinforcement learning (Sutton and Barto, 1998) was used to bias the selection of cases presented to those which candidates have done relatively poorly at solving.

While only a fraction of the training cases were seen by any individual candidate to provide the fitness measure used for determining whether a candidate became a parent for the next generation, the 100 best candidates (as well as a sample of the rest) were evaluated on the entire set of training cases as well as the out-of-set cases. The overall leader in each generation was the one that did best on the complete set of training cases. In addition, teams of the  $k$  best candidates in the population ( $2 \leq k \leq 30$ ) were evaluated over the complete set of cases, the team's prediction on each case being an unweighted average of the prediction of its members.

### 3 RESULTS

The overall results of the experiment are shown in Table 4, which shows the mean (with 95% confidence interval), median, maximum and minimum mean relative error found for each percentage level. The means and medians are over all of the runs in all of the subsets at each percentage. Since the same number of runs were done for each subset, the mean given is the same as the mean of the subset means, although the confidence interval is tighter.

#### 3.1 COMPARISON WITH PRIOR RESULTS

The "weighted average" column shows the mean of the subset means normalized over the number of cases in each subset. This is the relevant number to compare with the results in (Anderson, 2001, repeated in the table), as the results given for that experiment are the mean of the errors over all of the out-of-set cases.

Looking at the results, it is apparent that genetic programming does substantially better than hyperplane interpolation when trained on 40% and 60% of the data. The weighted means are outside the confidence intervals given, and a  $t$ -test shows that the means of both runs are significantly ( $P < 0.0005$ ) better than the hyperplane interpolation means. Looking at it another way, the hyperplane interpolation results require approximately 57% of the data to achieve the results GP can do with 40%. This represents a savings of nearly ten hours in data collection, compared with a run time averaging about eight and a half minutes. For the 60% level, the corresponding figures are 65% and four and a half hours.

Anderson did not report a number for the 20% level. To obtain his training sets, each element of the data set was chosen to be a training case with a probability equal to the level. Below 40%, it was deemed too likely that one of the tables would have too few training examples to be usable.

On the other hand, the GP results when training on 80% of the data are significantly worse than hyperplane interpolation. More work needs to be done to investigate why this is so, but as the focus of the work was to do well with small samples, the results are encouraging.

When compared against the analytical model presented in (Uysal, et al., 2001), with 40% of the data, the overall

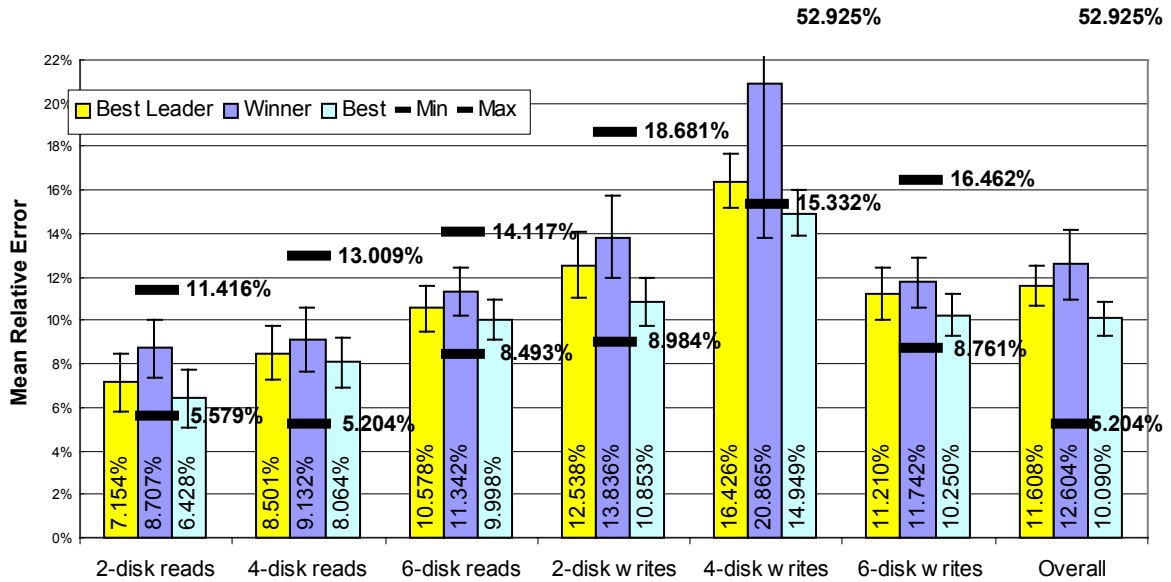


Figure 2 Out-of-set performance by subset, 40% case.

mean and the means of all of the subsets with the exception of 4-disk writes are less than the reported mean of 15%. Uysal, et al. validated their model on the 4- and 6-disk configurations using the same data, and with the same exception, over those subsets, only a single GP run produced an error greater than 15%. Thus, it appears that with this approach one can expect to generate models that have better prediction accuracy than those developed by hand by experts.

### 3.2 OVERFITTING

One striking feature of Table 4 is the presence of enormously bad values when training on 20% of the data. This brings up the everpresent danger of overfitting the training cases. In fact, out of 240 runs in the experiment, there was only one case of spectacularly overfitting the data. In the second run of the 6-disk read subset using 20% of the data, the winning candidate had a mean relative error of 7.45% over all the training cases—and 2,383,190,015% over the validation cases. In addition, there were two other runs with error greater than 100% (20% 4-disk read, run 3: 3,774% and 20% 2-disk read, run 6: 188%), and an additional eleven runs with error greater than 30%. All but one of these runs were when training on 20% of the data, where the total number of training cases ranged from 14 to 28. The only substantial overfitting seen at higher percentage levels was run 9 of the 4-disk write subset using 40% of the data, with an out-of-set error of 52.925%.

If the three outliers at the 20% level are removed, the mean on the remainder becomes  $21.932 \pm 6.669\%$ . Removing the next three ( $MRE > 60\%$ ) brings it down to  $19.221 \pm 4.218\%$ . In each of the six removed runs, an individual was found with out-of-set error  $< 35\%$ . (In

three cases  $< 20\%$ .) In addition, in three of the six runs, a “best so far” candidate had been identified with error less than 20%. This suggests that with more attention to the identification and selection of the winner, useful results should be expected with as few as 20% of the cases.

### 3.3 ROOM FOR IMPROVEMENT

Looking in depth at the 40% case, Figure 2 shows the out-of-set performance over the six subsets as well as overall performance. In each group, the center bar represents the candidate chosen as the winner, the left-hand bar represents the out-of-set best candidate that was chosen as leader at some point in the run, and the right-hand bar represents the candidate that had the best out-of-set performance during the entire run.<sup>1</sup> The heights of the bars represent the mean, and the whiskers indicate the 95% confidence levels. The other levels shown are the maximum and minimum values for the winner.

While the best individual seen, of course, cannot be used during a real run, the difference in height between the right and center bars is an indication of the efficacy of the method used for selecting the winner. At this percentage level, the winning candidate is, on average  $2.514 \pm 1.230\%$  worse than the best candidate seen. If the one bad ( $MRE=52.9\%$ ) run is removed, this number drops to  $1.913 \pm 0.360\%$ . So the method does reasonably well at choosing a good candidate, but there is some room for improvement. A  $t$ -test over all runs at the 40% level confirms that the best individual seen is significantly ( $P < 0.01$ ) better than the eventual winner.

<sup>1</sup> Strictly speaking, this is the candidate that had the best out-of-set performance of those fully tested. Only the top 100 and a small sample of others were tested each generation.

**Table 5** Mean Generation Found, 40% cases

Percent Used	Winner Found in Generation	Best Found in Generation	Best Leader Found in Generation
20%	287±6	160±23	125±21
40%	278±7	221±18	195±20
60%	277±7	247±13	229±16
80%	271±10	231±18	195±20

The difference in height between the left and center bars is an indication of the efficacy of the termination condition, as it indicates that the eventual winner is somewhat worse than one that would have been chosen had we stopped at some earlier point. Here the margin is  $1.536 \pm 1.183\%$  ( $0.948 \pm 0.269\%$  dropping the outlier) and the significance of the difference is  $P=0.054$ .

One other thing apparent from Figure 2 is that there is substantial difference between the performance on the various subsets, indicating that the problems posed are not equally easy to solve. A comparison of the means at the  $P<0.05$  level of significance shows that 2- and 4-disk reads are the easiest, followed by 6-disk reads and 6-disk writes. Next comes 2-disk writes, and finally, not surprisingly, 4-disk writes.<sup>2</sup>

## 4 ANALYSIS

We now turn to the question of the significance of the run

parameters.

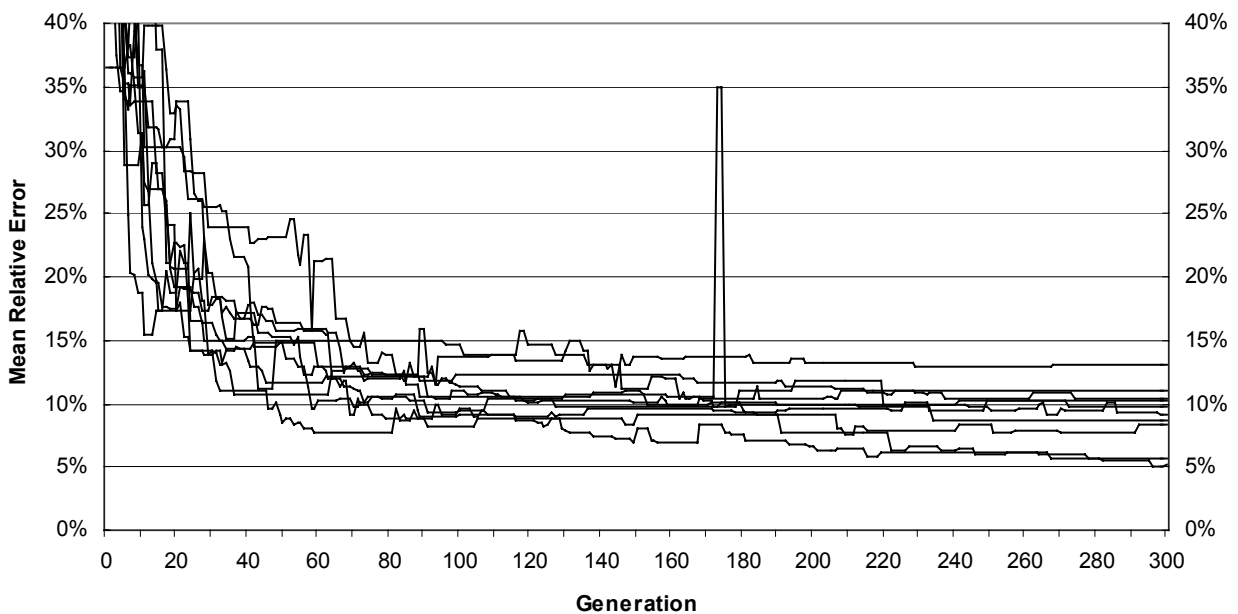
### 4.1 POPULATION SIZE

To investigate the sensitivity of the runs to population size, we ran 30 runs of each subset at the 40% level using a population of 1,000 candidates. The resulting overall mean relative error was  $15.220 \pm 1.108\%$ , a difference significant at the  $P<0.005$  level. Looking at the subsets individually, only the 4-disk write case (with the outlier in the original population) failed to show a significant increase in error. So this appears to be a problem that benefits from a larger population size.

### 4.2 LENGTH OF RUN

While large populations appear to be beneficial for this problem, an analysis indicates that our choice of 300 generations for the run length may have been overly conservative. At first glance, the numbers shown in Table 5 would seem to imply that the run is taking advantage of the full length of the run. Not only are the winners typically selected in the last thirty generations (in seven cases in the final generation), but the absolute best individual is typically found in the last third of the run. When we look at runs in detail, however, we see a different story.

Figure 3 shows the out-of-set mean relative error of the leader by generation for each of the runs at 40% on the 4-disk read subset. As is apparent in the figure, there is a precipitous drop in error for approximately the first eighty generations, but after that, the performance on most of the runs levels off, with the improvements, while continual,



**Figure 3** Out-of-set error for leader, 4-disk reads, 40%

<sup>2</sup> The mean for 6-disk writes is less than that for 2-disk writes at the  $P=0.053$  level.

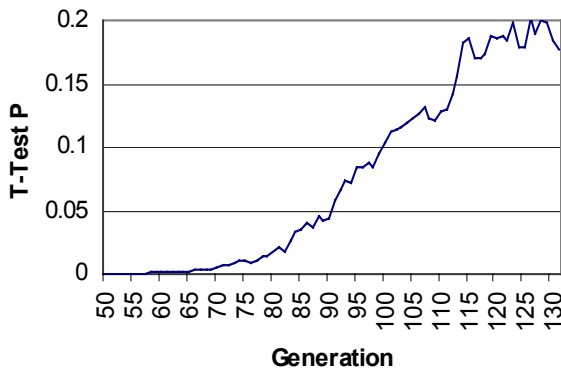


Figure 4 Improvement for 300-generation run vs.  $n$ -generation run

being slight. (It is heartening to see, however, that only one run made a very brief two-generation detour into seriously overfitting the data.)

If we perform  $t$ -tests comparing the final result to the result after each generation for the 40% case, as shown in Figure 4, we see that a 300-generation run is only significantly ( $P < 0.05$ ) better than runs of less than 91 generations. In another experiment in which the length of run was 1,000 generations (with varying numbers of runs per subset), a similar test shows that 1,000 generations is only significantly better than runs of less than 130 generations.

### 4.3 AGGREGATION

One somewhat unusual technique used is the aggregation of good members of the population into teams that vote on the final solution. In this experiment, we only used the simplest of GPLab’s aggregation mechanisms, in which all sets of the  $k$  best candidates in a population (for  $k$  up to 30) are tested as a team. It should be noted that the aggregation is for purposes of selecting winners only—the aggregates are not considered to be parents, and, therefore, there is no evolution of teams *per se*.

In the 40% runs, the winner selected is a team 71.7% of the time (43 runs). Figure 5 shows the distribution of the sizes of winning teams. The intuition here is that, rather than converging on a single winner, there are often competing “subspecies”, each nearly the best, jockeying for the first place position. While the winner selection method discussed in section 4.4 removes some of the problems that this can cause, especially due to overfitting, aggregation can allow solutions which are combinations of candidates from subspecies which are each good at solving cases in different regions of the input space.

In another set of runs, we asked GPLab to use hill-climbing to add and remove good candidates to and from the team being constructed. While this occasionally resulted in leaders being chosen during the run, it did not appear that any final winners were constructed using hill-climbing.

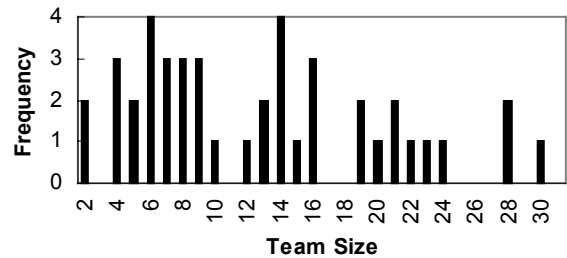


Figure 5 Frequency of team size, 40% winners

### 4.4 WINNER SELECTION

Another unusual aspect of our setup is the separation of the fitness measures used for parent selection and winner selection. Parents were chosen as usual based on the performance on fitness cases seen during the training phase, but winners were selected based on overall performance of the candidates on all of the training cases.<sup>3</sup> This, combined with dynamic fitness case selection, was an attempt to avoid the overfitting that so often happens in genetic programming runs, in which a lucky individual or overfitting subpopulation happens to occupy the top spot at the end of the run and gets selected as the winner.

The technique appears to be useful. For the 40% runs, while the winner selected on the basis of all of the training cases had a mean error of  $12.604 \pm 1.615\%$  on out-of-set cases, the candidate that did best during its training phase had an out-of-set mean error of  $18.012 \pm 5.410\%$ , for a net gain of  $5.408 \pm 5.450\%$ . The largest difference was 157.724%, and there were four cases out of sixty in which the training-best candidate had better out-of-set performance than the winner. That such winners should be easy to spot is evidenced by the fact that over all of the training data, winners outperformed training-best candidates by  $9.261 \pm 5.879\%$ .

The picture is a bit less clear-cut when outliers are removed. The one overfitting (out-of-set MRE = 52.925%) winner was outperformed by its training-best runmate by 25.841%. On the other hand, there were only two substantially overfitting training-best candidates (out-of-set MREs = 169.504% and 61.890%) respectively. If these runs are removed, the out-of-set difference shrinks to  $2.445 \pm 0.683\%$ , still significantly ( $P < 0.001$ ) better, but a less spectacular improvement.

In early runs, we experimented with GPLab’s *tripartite fitness case division*, in which the fitness cases are partitioned into a training set, a testing set, and a validation set. This last set, which must be considered “in-set data” for purposes of comparison, contains cases not used for purposes of selecting parents, but which may be used to control other run parameters during the run, including the decision to terminate and the selection or construction of a winner. We found that using such a

<sup>3</sup> But not, of course, the testing cases.

validation set for winner selection appeared to be better than using the training set—but not as good as simply enlarging the training set. That is, in runs with 40% training cases and 20% validation cases, the winner selected on the basis of validation fitness tended to outperform the winner selected on the basis of training fitness, but it appeared to not do as well as the winners selected on the basis of training fitness from runs which used 60% of the cases as training cases. We have not yet done enough runs to say this with any confidence, however.

#### 4.5 SAMPLE SIZE

We now consider the impact of the size of the data set. One unexpected result, reported in Table 4, was that while the system did better when training on 60% of the data than it did with 40% of the data, things actually got worse when the sample size was increased to 80%. Checking significance levels backs this up. While it is significantly ( $P < 0.001$ ) better to have 40% of the data than 20% and ( $P < 0.01$ ) better to have 60% of the data than 40%, when comparing the significance levels of the 80% and 60% runs, we do no better than  $P = 0.21$ . Indeed, when comparing the 80% runs to the 40% runs, the significance is reduced to  $P = 0.04$ .

This is hard to account for, but it may be due to the increased likelihood that the now-smaller testing set is not representative of the space as a whole rather than some sort of interference from the now-larger training set.

### 5 FUTURE WORK

While the initial results are promising and bring the prediction accuracy into the region which makes it useful for design systems like Minerva, more work needs to be done to drop the error rate to the desired 3–5% range. Since population size appears to be a significant factor, an obvious avenue to explore is the use of even larger populations, perhaps recouping some of the time by shortening the runs, as runs of longer than about 125 generations appear to be unnecessary.

Another question unexplored in this paper is the impact of the operator set on the mechanism's performance. Given the large range of data and the fact that powers of two tend to be important to computer systems, one operator that appears to be useful, although we have not yet done enough runs to be able to say anything statistically sound, is a binary logarithm, perhaps restricted to taking its argument from the set of input variables.<sup>4</sup> Other likely wins in this area are the addition of Boolean and relational operators, along with conditional branches, allowing the problem to be split into separate sub-cases.

---

<sup>4</sup> The type algebra of GPLab allows the addition of arbitrary *attributes* to types, so one can, for example, talk about “Boolean input variables” and “Real-valued constants”. This extends to the return values, so one can include an addition operator that takes two constant integers and returns a constant integer.

Aggregation appears to be a promising technique, and we are investigating several mechanisms, including some that take into account candidates from prior generations. We are also planning on doing experiments to test the sensitivity to various other runtime parameters controlling dynamic fitness case selection, evaluation, reproduction, and the decision to terminate a run.

## 6 CONCLUSIONS

Genetic programming appears to be a promising technique for modeling disk arrays. In this paper we have shown that it is capable of producing models that outperform both a published analytical model and, for small amounts of input data, a currently-used interpolation-based approach, models which have acceptable, if not outstanding error behavior.

A statistical analysis indicates that the problem is sensitive to population size but not run length (for runs longer than about 125 generations) and lent support to the use of aggregation and the method used for selecting the overall winner. While there is still some danger of overfitting, this is less common than might be expected and appears to be helped by the combination of dynamic fitness case selection and winner selection used, although this was not proven.

## ACKNOWLEDGEMENTS

We would like to thank Eric Anderson for his help in posing the problem, for helping us acquire and understand the data, and for his comments on an earlier draft. We would also like to thank Jaap Suermondt for his comments and Jerry Shan for his help in understanding the statistics.

## References

- Alvarez, Guillermo A.; Borowsky, Elizabeth; Go, Susie; Romer, Theodore H.; Becker-Szendy, Ralph; Golding, Richard; Merchant, Arif; Spasojevic, Mirjana; Veitch, Alistair; and Wilkes, John. 2001. Minerva: an automated resource provisioning tool for large-scale storage systems. Technical Report HPL-2001-139, Hewlett-Packard Laboratories, June, 2001. To appear in *ACM Transactions on Computer Systems*.
- Anderson, Eric. 2001. Simple table-based modeling of storage devices. Hewlett-Packard Laboratories Storage Systems Project technical memo HPL-SSP-2001-4. July 14, 2001.
- Koza, John R. 1990. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*. Stanford University Computer Science Department Technical Report STAN-CS-90-1314. June, 1990.
- Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, MIT Press.

- Montana, David J. 1995. Strongly typed genetic programming. *Evolutionary Computation* 3(2):199–230.
- Sutton, Richard S; and Barto, Andrew G. 1998 *Reinforcement Learning: An Introduction*. Cambridge, MA, MIT Press.
- Uysal, Mustafa; Alvarez, Guillermo A.; and Merchant, Arif. 2001. A modular, analytical throughput model for modern disk arrays. *Proceedings of the Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS-2001)*, p. 183–192, August 15–18, 2001, Cincinnati, OH.