

## Pyramid Search: Finding Solutions for Deceptive Problems Quickly in Genetic Programming

**Vic Ciesielski**

School of Computer Science  
and Information Technology  
RMIT University, GPO Box 2476V  
Melbourne Victoria 3001, Australia  
vc@cs.rmit.edu.au

**Xiang Li**

School of Computer Science  
and Information Technology  
RMIT University, GPO Box 2476V  
Melbourne Victoria 3001, Australia  
xiali@cs.rmit.edu.au

**Abstract-** In deceptive problems many runs lead to sub-optimal solutions and it can be difficult to escape from these local optima and find the global best solution. We propose a pyramid search strategy for these kinds of problems. In the pyramid strategy a number of populations are initialised and independently evolved for a number of generations at which point the worst performing populations are discarded. This evolve/discard process is continued until the problem is solved or one population remains. We show that for a number of deceptive problems the pyramid strategy results in a higher probability of success with fewer evaluations and a lower standard deviation of the number evaluations to success than the conventional approach of running to a maximum number of generations and then restarting.

### 1 Introduction

One of the major issues in genetic programming is the management of the evolutionary process. In most problems it is desirable to get solutions with the smallest possible amount of computation. Unfortunately many parameters and search strategies are possible and finding the best combination of parameter values and search strategies for any given problem is very difficult. Furthermore, some problems appear to be inherently difficult for genetic programming in that they are deceptive and lead to premature convergence [12, p191]. In general this means that during the course of evolution, programs are generated which appear to be quite good but continued evolution might not lead to improved programs. These programs are only local optima. In order to find better programs it is necessary to restart the evolutionary process from the beginning with a new random initial population and hope that the new run will result in better programs. This paper describes an approach to managing this process in a way that generally leads to a higher probability of success with a smaller number of fitness evaluations.

In the early days of genetic algorithms and genetic programming it was generally assumed that large populations were preferable because they maximised diversity and

hence would give a higher chance of avoiding premature convergence and finding the best solution [11]. However, more recent work has shown that large population sizes are not necessarily best and that solutions can often be found with a much smaller number of evaluations using small populations [14, 16]. For example, 10 runs with a population of 100 could give a higher chance of success than 1 run of 1000.

There have been many approaches to the premature convergence problem in genetic algorithms and genetic programming. Generally these have investigated different ways of maintaining diversity in the population, based on the assumption that the process is less likely to be trapped in a local optimum if there is a large variety of different individuals in the population. Some investigations have targeted the crossover operation [1, 9], some the selection operation [2, 3, 18] and some, like ourselves, have investigated population and search management [5, 6, 8, 17, 19]. Two approaches which specifically target population/search management are the grid model (fine grained parallelism) and the island model (coarse grained parallelism). In the grid model the population is viewed as a two dimensional array. An individual can only recombine with a partner within a fixed neighbourhood [17]. In the island model a number of individual populations are evolved independently and periodically exchange genetic information[8]. Unfortunately in these models diversity maintenance generally leads to a larger number of evaluations in each run.

Loveard [14, 15], working with classification problems, found that some runs were “predisposed” to success and some to failure, that is, whether a run succeeded or not generally depended on the genetic mix in the initial population and that it was generally possible to tell after a relatively small number of generations whether a run would lead to a good fitness. This occurred for both large and small populations and that continuing the run for many generations only sometimes resulted in a bad run eventually coming good. He proposed computational strategies in which a number of populations were initialised, not just one. These populations were evolved independently in parallel. During the

course of the evolution populations which appeared to be predisposed to failure were either terminated or replaced by copies of populations that appeared to be predisposed to success. Loveard showed that, for the same total number of evaluations, the accuracy of the evolved classifiers could be improved by terminating bad populations early and redirecting computational effort to the better populations. One could view these strategies as giving a better return on the number of evaluations performed than the standard strategy. As an example, a standard strategy might require 40 runs of a population of 100 for a maximum of 50 generations giving a total of  $40 \times 100 \times 50 = 200,000$  evaluations. Loveard showed that if these 200,000 evaluations were organised according to his proposals, then more accurate classifiers could be evolved for the same computational effort.

The primary question we seek to address is somewhat different to the one addressed by Loveard. Rather than looking for the best solution possible in some given number of evaluations, we would like to find ways of organising the computation to get the (or an acceptable) solution with as small a number of evaluations as possible. This can be viewed as maximising the probability of getting a solution as early in the evolutionary process as possible.

## 1.1 Hypotheses

Our specific hypotheses are:

1. A ‘pyramid’ strategy of initialising multiple populations, evolving them in parallel and periodically eliminating the worst population gives a good probability of success within a relatively small number of evaluations.
2. For any problem there is an optimal shape of the pyramid (wide/narrow base, tall/short height).
3. The pyramid approach is superior to the standard approach and the fine grained approach in that it leads to solutions in a smaller number of evaluations.

Since it requires a number of populations to be used at the outset, the pyramid approach is not expected to give any benefits to ‘easy’ problems which can be solved in a small number of generations with small populations. Thus we have selected problems which are known to be deceptive and problems which are known to be difficult in that they require a lot of computation. We examine the performance of the pyramid strategy on the Max problem, a 3 variable symbolic regression problem, the 5 bit even parity problem and a difficult object detection problem.

We employ a number of metrics to compare performance of various strategies. These are (1) plots of probability of success vs number of evaluations, (2) average and standard deviation of the number of evaluations to the first solution,

*pyramid(np, ps, pr, ng)*

*np* Number of populations

*ps* Population Size

*pr* Pruning ratio, Fraction of remaining populations to remove

*ng* Number of Generations between prunings

Initialise *np* populations of size *ps*

*popsleft* = *np*

Until problem solved or a single population remains

do

Evolve each population for *ng* generations

Remove the  $\text{round}(pr * \text{popsleft})$  least fit populations

$\text{popsleft} = \text{popsleft} - \text{round}(pr / \text{popsleft})$

end do

If problem not solved

then continue evolving single population until

problem solved or maximum evaluations is reached

Figure 1: The Pyramid Algorithm

and (3) number of evaluations needed to give a 99% probability of success. In the experiments all of these are estimated from 100 runs.

## 2 The Pyramid Strategy

The basic pyramid algorithm is shown in Figure 1.

Figure 2 gives a picture of the evolutionary process for a pyramid strategy where the population size (*ps*) is 50, the number of populations (*np*) is 10, the pruning ratio (*pr*) is 0.2 and the number of generations between prunings (*ng*) is 5. In the initialisation step 10 populations of size 50 are randomly generated. Each population is then evolved independently for 5 generations. At this stage the populations are ranked based on the fittest individual in each population. The population with the best fittest individual will be ranked highest while the population with the worst best individual will be ranked lowest. The  $\text{round}(0.2 \times 10) = 2$  lowest ranking populations will then be deleted. The remaining 8 populations are then evolved independently for another 5 generations when the worst  $\text{round}(0.2 \times 8) = 2$  populations are removed. This process continues until the problem is solved or until a maximum number of generations is reached. In the example, one population remains after 40 generations and evolution continues with this one population until 100 generations are completed.

The basic intuition behind this strategy is as follows: Some of the 10 initial populations will be predisposed to be good, some bad [14, 15]. This is illustrated in figure 3. There are runs for all three population sizes that converge

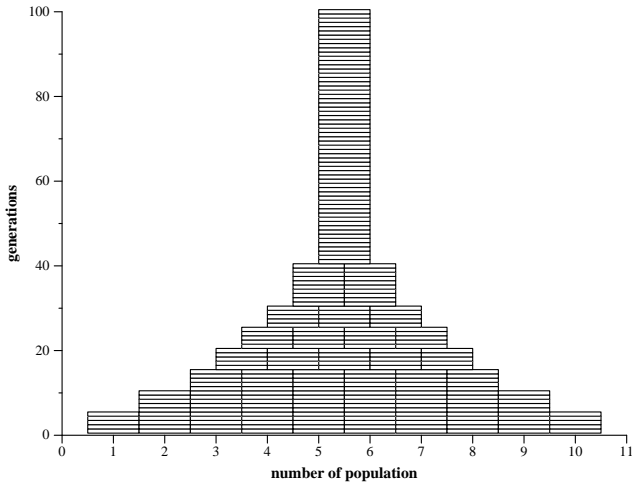


Figure 2: Example of the Pyramid Strategy

quite quickly. There are also many runs which have reached a plateau. Some of these eventually come good, but many are still stuck at over 2,225,000 evaluations. This predisposition is often evident early in the number of generations. Since bad runs usually stay at a plateau for a while before improving, or don't actually improve at all, the bad populations can be deleted and effort invested in the better populations. By the time one population is left it should be one that is predisposed to success. For problems which are particularly deceptive or difficult the base of the pyramid can be made wider and/or the pruning ratio decreased and/or the number of generations between prunings increased.

In choosing the test problems we have tried to maximize the variety of problem types. Three are classic genetic programming problems [12, 13] and one is a very difficult real world object detection problem. Population sizes have been chosen to be small, medium and large. The other parameter values were chosen by preliminary experimentation.

### 3 The Max Problem

This problem has a function set of  $\{+\times\}$  and a terminal set of  $\{0.5\}$ . The objective is to generate a program tree that maximises the value obtained by evaluating the program tree for a given maximum depth. The desired solution is unique and is a full tree with 0.5 at each terminal, + functions at the two lowest levels of the tree and  $\times$  functions at all the other levels. This is a known deceptive problem which displays obvious premature convergence behaviour [4, 13]. We have used a depth of 6 in our experiments.

Figure 4 shows the probability of success vs number of evaluations for a single population of size 49, Norm49, a single population of size 490, Norm490, and a pyramid search with 10 initial populations of size 49 and a prun-

ing ratio of 0.2. Points on these graphs were generated by running each strategy 100 times and noting the number of evaluations to the first solution of each run and using a histogram of width 4,900 to estimate the probability density functions shown in the figure. In this figure  $ng$ , the number of generations between pruning operations is 5. Early experimentation suggested that 5 was a good value for this parameter and unless otherwise stated all results presented are for  $ng = 5$ . Our original intention was to include grid and island methods in our comparisons, hence our choice of  $7 \times 7 = 49$  for population size. However, the grid approach performed poorly in comparison with the other approaches (Figure 6) and we did not have access to an implementation of the island model so we did not pursue this. The line for Norm49 does not begin at 0 as there were some successes in the first 4,900 evaluations. Note that the pyramid method has a very high probability of success within the first  $4 \times 4,900$  evaluations. Furthermore the variation in how long the runs take is quite small compared to the two other methods (also clear from table 1). Essentially all pyramid runs have finished successfully by  $4 \times 4,900$  evaluations while the other 2 methods still have runs completing at  $25 \times 4,900$  evaluations.

Figure 5 shows the same data as Figure 4 but as a cumulative probability distribution. The pyramid method is virtually guaranteed to find a solution within  $3 \times 4,900$  evaluations.

An alternative way to compare the methods is to look at the average number of evaluations to the first solution. This data is shown in Table 1. (Note, data for 2 additional pyramids, not included in figures 4 and 5, are given here and in table 2.) The 'Num Suc' column is the number of successes out of the 100 runs. A  $t$ -test comparing average number

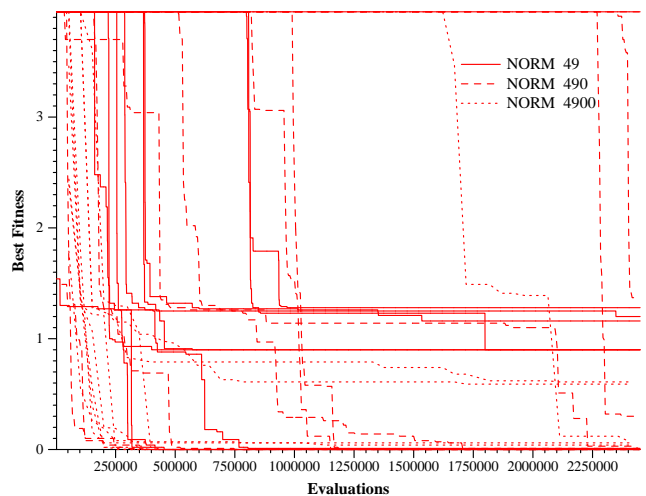


Figure 3: Ten Randomly Selected Runs for Each Population Size, Symbolic Regression Problem

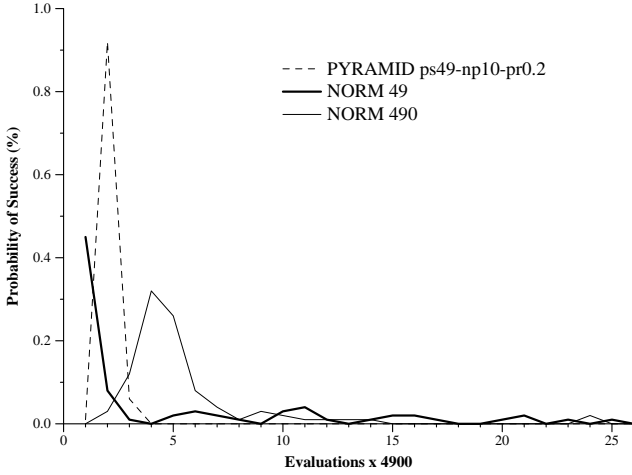


Figure 4: Probability of success, Max Problem

of evaluations to the first solution shows that pyramid Pyr-ps49-np10-pr0.2 is significantly better than Norm49 and Norm490 at the 0.0001 level. While the means and standard deviations are not directly comparable since the non converging runs are ignored, two of the pyramid methods have a very high success rate, a significantly lower mean and an amazingly small standard deviation compared to the two normal methods.

A further way to compare the methods is to look at the number of evaluations necessary to reach, say, a 99% probability of finding the solution. This would usually involve running a strategy multiple times. For  $R$  runs the probability of success in at last one run of  $i$  evaluations is given by  $1 - [1 - P(i)]^R$  [11, p193] where  $P(i)$  is the probability of success of a single run after  $i$  evaluations. Of course, for comparison purposes,  $R$  runs would involve  $i \times R$  eval-

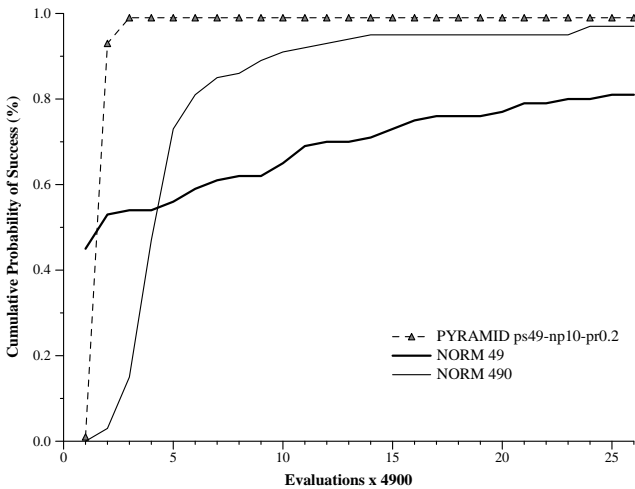


Figure 5: Cumulative Probability of success, Max Problem

Methods	Num Suc	Mean	SD.
Norm49	81	21740	30838
Norm490	97	24361	16870
Pyr-ps49-np10-pr0.1	100	11253	1675
Pyr-ps49-np10-pr0.2	99	7961	1194
Pyr-ps49-np10-pr0.5	94	6885	11099

Table 1: Mean and Standard Deviation, Number of Evaluations to First Solution, Max Problem

Method	Runs	Total Evals
Pyr-ps49-np10-pr0.2	1 for 3x4,900	3x4,900
Pyr-ps49-np10-pr0.1	1 for 4x4,900	4x4,900
Pyr-ps49-np10-pr0.5	8 for 3x4,900	8x4,900
Norm49	9 for 1x4,900	9x4,900
Norm490	4 for 6x4,900	24x4,900

Table 2: Runs and Evaluations to Reach a Probability of Success of 99%, Max Problem

uations. Ideally one would choose the point of diminishing returns as the time to terminate a run and restart. We have identified this point for each method by a process of searching through the possibilities using the data plotted in figure 5. The smallest number of evaluations needed for the Norm49 method to reach a 99% probability of success was 9x4,900 which is obtained by 9 runs for 1x4,900 evaluations each. Table 2 shows the results of this analysis. This table compares the methods on the assumption that the user would know the optimal point to terminate a run and start again.

Clearly the pyramid method would be the preferred method if the goal is to have the best chance of a solution with as few evaluations as possible.

### 3.1 Other Pyramid Shapes

The previous section considered a pyramid of one shape only. In this section we consider some variations in the shape of the pyramid. Figure 6 shows the results of runs with different pruning ratios. Results for a grid of size 7x7 with a neighbourhood of size 3 are also presented for comparison. Pyramids with different values of  $np$  are explored for the symbolic regression problem.

Analysis of different shaped pyramids reveals the limits of what can be expected from the pyramid strategies. A pyramid with a small  $np$  will tend to behave like a run on a single population. If  $np = 1$  then the pyramid degenerates into a single run with a population size of  $ps$ . For example, it can be seen in figure 8 that the curve for pyramid pyr-ps49-np10-pr0.5, which has a high pruning ratio and degenerates into a single population after 20 generations, is quite close to the curve for Norm 49. A pyramid that has a large

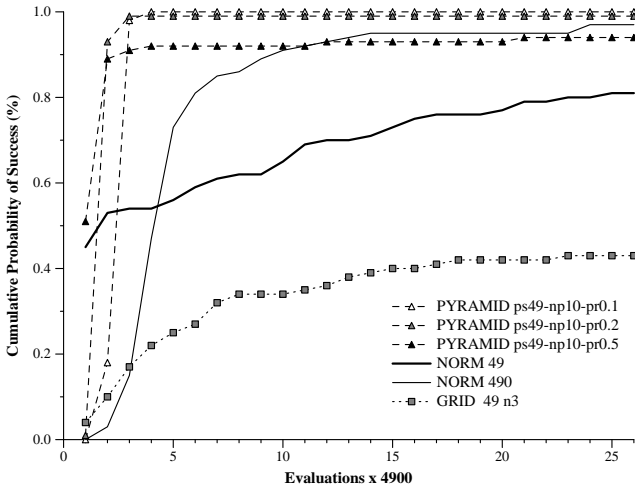


Figure 6: Cumulative Probability of success, Max Problem

$np$  and a small  $pr$  should behave like  $np$  runs of a population of size  $np$ . If  $pr = 0$  then the pyramid turns into  $np$  normal runs with a population size of  $ps$ . This, together with the empirical results of figure 6, suggests that there will be a set of pyramid parameters that are optimal for any given problem.

#### 4 Symbolic Regression Problem

For this problem, 50 quadruples of values were randomly extracted from the equation  $w = zy - (0.24x)/y^2$ . The task is to evolve an expression which gives a small error when evaluated at these 50 quadruples. Fitness evaluation thus involves evaluating an evolved program 50 times. The hope is that the original equation will be evolved, although

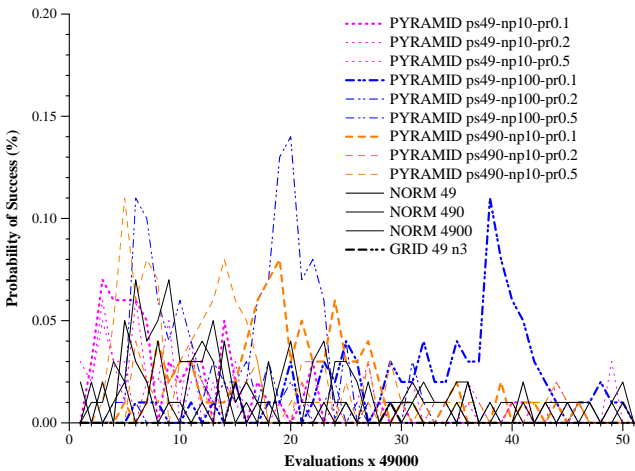


Figure 7: Probability of success, Symbolic Regression Problem

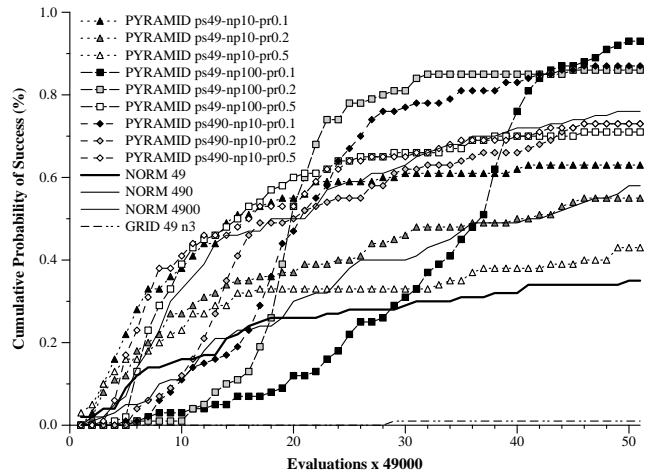


Figure 8: Cumulative Probability of success, Symbolic Regression Problem

this rarely happens. Success is achieved when the error drops beneath a predefined threshold. The difficulty of the problem can be artificially manipulated by the choice of threshold. We have used a threshold of 0.01 which gives a considerably difficult problem. The functions used are  $\{+, -, \times, /\}$  and the terminals  $\{x, y, z, rand\}$ . For this problem,  $ng$ , the number of generations between prunings was 50.

We have performed a similar analysis to the one described in the previous section for the symbolic regression problem. The probability of success is shown in figure 7. The corresponding cumulative probabilities are shown in figure 8.

While there is a considerable amount of clutter in both of these figures a number of things do stand out. In figure 7 the pyramid methods are associated with sharp spikes whereas the normal methods have a generally low probability of success spread over a large number of evaluations. The three pyramids with  $ps = 49$  and  $np = 100$  stand out with large clear peaks. Inspection of figure 8 reveals that the curves for the pyramid methods tend to bulge towards the upper right. This indicates that the probability of success is higher with fewer evaluations. Also, the top 3 methods at 51x49,000 evaluations are pyramid methods.

The same general patterns as for the Max problem are repeated here.

Table 3 shows the number of successful runs out of 100 and the means and standard deviations of the successful runs. There are pyramids which have higher success rates and lower standard deviations than the normal runs as was the case for the Max problem, however the differences in standard deviations is nowhere near as striking.

Table 4 confirms that, if the goal is find a solution as soon

Methods	Num Suc	Mean	SD
Norm49	35	743706	630238
Norm490	58	1084244	665775
Norm4900	76	801346	586208
Pyr-ps49-np-10-pr0.1	63	491253	418212
Pyr-ps49-np-10-pr0.2	56	787681	679438
Pyr-ps49-np-10-pr0.5	43	716362	750473
Pyr-ps49-np-100-pr0.1	96	1634739	527168
Pyr-ps49-np-100-pr0.2	90	1059778	499312
Pyr-ps49-np-100-pr0.5	72	651165	482492
Pyr-ps490-np-10-pr0.1	91	1084939	595178
Pyr-ps490-np-10-pr0.2	76	973723	622945
Pyr-ps490-np-10-pr0.5	75	700088	602952

Table 3: Mean and Standard Deviation, Number of Evaluations to First Solution, Symbolic Regression Problem

as possible, the pyramid methods are preferred. For this problem it appears that a large number of small populations is best.

## 5 5-Bit Even Parity Problem

In this task it is necessary to generate a 1 or 0 depending on whether the number of ones in a 5 bit binary string is odd or even [11, p529]. For example the string 01100 requires 0 while 11010 requires 1. The function set is {AND, OR, NAND, NOR } while the terminal set is {S1,S2,S3,S4,S5} where  $S_i$  is the value of the  $i$ th digit. There are  $2^5 = 32$  fitness cases. This is known to be difficult and deceptive and has been the subject of a number of studies, for example [10, 20]. The difficulty of the problem can be artificially manipulated by adjusting the desired error rate. An error rate of 0 out of 32 would be a perfect solution. We have used 9.

Methods	Runs	Total Evals
Pyr-ps49-np-100-pr0.2	4 for 27x49,000	108x49,000
Pyr-ps49-np-100-pr0.5	7 for 17x49,000	119x49,000
Pyr-ps49-np-10-pr0.1	8 for 15x49,000	120x49,000
Pyr-ps490-np-10-pr0.1	5 for 26x49,000	130x49,000
Pyr-ps490-np-10-pr0.5	8 for 17x49,000	136x49,000
Pyr-ps49-np-10-pr0.5	35 for 4x4,9000	140x4,9000
Pyr-ps49-np-100-pr0.1	3 for 48x49,000	144x49,000
Pyr-ps490-np-10-pr0.2	7 for 23x49,000	161x49,000
Norm4900	7 for 23x49,000	161x49,000
Norm49	32 for 7x49,000	224x49,000
Pyr-ps49-np-10-pr0.2	7 for 45x49,000	315x49,000
Norm490	7 for 46x49,000	322x49,000

Table 4: Runs and Evaluations to Reach a Probability of Success of 99%, Symbolic Regression Problem

Figure 9 and tables 5 and 6 give the results for this problem.

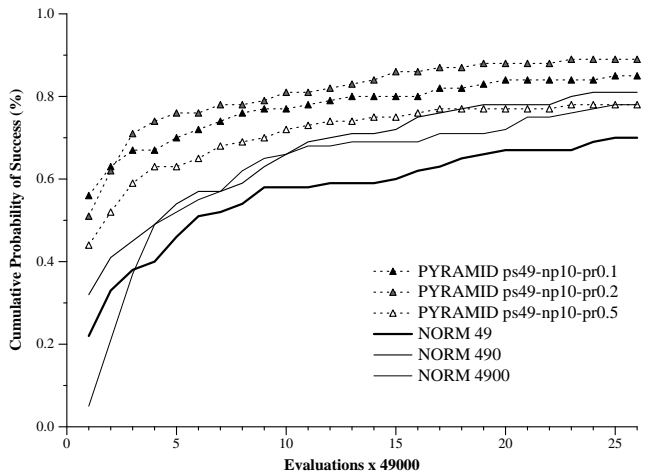


Figure 9: Cumulative Probability of success, 5-Bit Even Parity Problem

## 6 Object Detection Problem

This problem comes from a computer vision application in orthodontics where a number of landmarks in facial X-rays need to be found as part of the treatment planning process [7]. Programs which use an input window of pixels of around  $40 \times 40$  pixels are evolved. Fitness evaluation involves sweeping the evolved program over a larger picture of  $70 \times 85$  pixels and calculating the detection and false alarm rate. The number of generations between prunings was 50. The probabilities in figure 10 and tables 7 and 8 were estimated from 50 runs. The genetic programming approach proved to be very accurate in detecting a number of the landmarks, however the fitness evaluation is very computationally expensive and the evolutionary runs can take 5-10 hours. Using the pyramid strategy has reduced this by about 30%.

On all four problems the patterns and results for the pyramid strategy are very consistent. The graphs of probability

Methods	Num. Suc.	Mean	SD.
Norm49	70	261868	319034
Norm490	81	246299	296558
Norm4900	78	277069	295567
Pyr-ps49-np10-pr0.1	85	136129	239801
Pyr-ps49-np10-pr0.2	89	131332	224901
Pyr-ps49-np10-pr0.5	78	134411	213378

Table 5: Mean and Standard Deviation, Number of Evaluations to First Solution, 5-Bit Even Parity Problem

Methods	Runs	Total Evals
Pyr-ps49-np-10-pr0.1	7 for 1x49,000	7x49,000
Pyr-ps49-np-10-pr0.2	8 for 1x49,000	8x49,000
Pyr-ps49-np 10-pr0.5	9 for 1x49,000	9x49,000
Norm490	13 for 1x49,000	13x49,000
Norm49	20 for 1x49,000	20x49,000
Norm4900	8 for 4x49,000	32x49,000

Table 6: Runs and Evaluations to Reach a Probability of Success of 99%, 5-Bit Even Parity problem

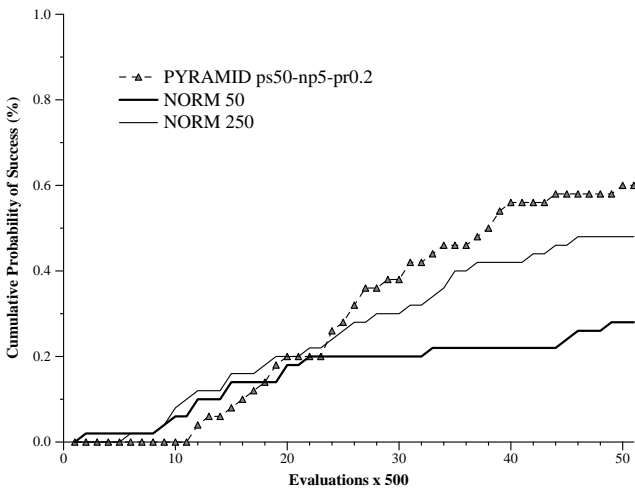


Figure 10: Cumulative Probability of success, Object Detection Problem

vs number of evaluations all have pyramids giving a higher probability of success with fewer generations than the usual approach. The success rate of some pyramids is always higher, the standard deviation of the number of evaluations to the first success is generally smaller and the number of evaluations to reach a 99% probability of success is smaller. For each problem we computed the average size of the population in each pyramid and performed a NORM run with this population size. In all cases this was inferior to the best pyramids.

## 7 Determining Pyramid Shape

We have discovered 10 populations of size around 50, a pruning ratio of 0.2 and a number of generations between prunings of 5 to be a good set of defaults. If the problem is difficult, then the pruning ratio can be made smaller or the number of populations, and perhaps population size, increased. If the default parameters solve the problem comfortably then a more aggressive pruning ratio up to 0.5 can be used.

Methods	Num Suc	Mean	SD
Norm50	14	10865	7201
Norm250	24	11868	5889
Pyr-ps50-np5-pr0.2	32	14103	5803

Table 7: Mean and Standard Deviation, Number of Evaluations to First Solution, Object Detection Problem

Methods	Runs	Total Evals
Pyr-ps49-np-5-pr0.2	7 for 39x500	273x500
Norm250	10 for 37x500	370x500
Norm50	229 for 2x500	458x500

Table 8: Runs and Evaluations to Reach a Probability of Success of 99%, Object Detection Problem

## 8 Conclusions

We have investigated a way of organising the evaluations in a genetic programming run in order to maximise the probability of success with as few evaluations as possible. The pyramid search strategy offers a variety of ways of managing a run. Some of these are better than others. The best pyramid strategies result in considerable improvement over the standard approach of multiple runs to some maximum number of generations. Furthermore, the standard deviations of the number of evaluations to success are lower, in the case of the Max problem, strikingly so.

Our experiments have been carried out on a single processor. Efficient implementation of pyramid search on a multiprocessor environment is left for further work.

We have shown, through a combination of analysis and empirical evidence, that there is likely to be an optimal pyramid shape for a problem for a given population size. We have not attempted to find the optimal shape, however we have established that there are a large number of shapes that outperform the standard strategy. While the pyramid method has the disadvantage of introducing a number of new parameters, it is relatively easy to find parameters that work well. We could summarise our results as: “You need to do multiple runs anyway to get a solution so you may as well do them the pyramid way and not waste time on the bad ones”.

## Acknowledgements

This work was partially supported by grant EPPNRM054 from the Victorian Partnership for Advanced Computing. We thank Andrew Innes and Jeff Campbell for performing the object detection runs and Brian Lam and Andy Song for helpful discussions.

## Bibliography

- [1] Zeke S. H. Chan, H. W. Ngan, and A. B. Rad. A new method to resist premature convergence: Synchronising gene-convergence with correlated recombination. In Scott Brave and Annie S. Wu, editors, *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, pages 74–79, Orlando, Florida, USA, 13 July 1999.
- [2] Vic Ciesielski and Dylan Mawhinney. Prevention of early convergence in genetic programming by replacement of similar programs. In Xin Yao, editor, *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 1, pages 67–72, Honolulu, May 2002. IEEE.
- [3] Larry J. Eshelman and J. David Schaffer. Preventing premature convergence in genetic algorithms by preventing incest. In Lashon B. Belew, Richard K.; Booker, editor, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 115–122, San Diego, CA, July 1991. Morgan Kaufmann.
- [4] Chris Gathercole and Peter Ross. An adverse interaction between crossover and restricted tree depth in genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 291–296, Stanford University, 1996. MIT Press.
- [5] Jun He and Xin Yao. From an individual to a population: an analysis of the first hitting time of population-based evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):495–511, October 2002.
- [6] Jianjun Hu and Erik D. Goodman. The hierarchical fair competition (HFC) model for parallel evolutionary algorithms. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 49–54. IEEE Press, 2002.
- [7] Andrew Innes, Vic Ciesielski, John Mamutil, and Sabu John. Landmark detection for cephalometric radiology images using genetic programming. *International Journal of Knowledge Based Intelligent Engineering Systems*, page [To Appear], 2003.
- [8] Makoto Iwashita and Hitoshi Iba. Island model GP with immigrants aging and depth-dependent crossover. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 267–272. IEEE Press, 2002.
- [9] Maarten Keijzer, Conor Ryan, Michael O’Neill, Mike Catolico, and Vlado Babovic. Ripple crossover in genetic programming. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP’2001*, volume 2038 of LNCS, pages 74–86, Lake Como, Italy, 18-20 April 2001. Springer-Verlag.
- [10] Emin Erkan Korkmaz and Göktürk Üçoluk. Controlled genetic programming search for solving deceptive problems. In Erick Cantú-Paz, editor, *Late Breaking Papers at the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 295–300, New York, NY, July 2002. AAAI.
- [11] John R. Koza. *Genetic Programming: on the Programming of Computers by means of Natural Selection*. The MIT Press, 1992.
- [12] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.
- [13] W. B. Langdon and R. Poli. An analysis of the MAX problem in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second International Conference on Genetic Programming*. Morgan Kaufmann, July 1997.
- [14] Thomas Loveard. Genetic programming with meta-search: Searching for a successful population within the classification domain. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Proceedings of the 6th European Conference on Genetic Programming (EuroGP 2003)*, pages 119–129, Berlin, 2003. Springer.
- [15] Thomas Loveard and Vic Ciesielski. Genetic programming for classification: An analysis of convergence behaviour. In Bob McKay and John Slaney, editors, *Proceedings of the 15th Australian Joint Conference on Artificial Intelligence (AI2002), Lecture Notes in Computer Science 2557*, pages 309–320, Canberra, December 2002. Springer.
- [16] Sean Luke. When short runs beat long runs. In Lee Spector et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 74–80, San Francisco, California, USA, 2001. Morgan Kaufmann.
- [17] Bernard Manderick and Piet Spiessens. Fine-grained parallel genetic algorithms. In *International Conference on Genetic Algorithms, ICGA’89*, pages 428–433, 1989.
- [18] Conor Ryan. Pygmies and civil servants. In Kenneth E. Kinneer, Jr., editor, *Advances in Genetic Programming*, chapter 11. MIT Press, 1994.
- [19] Hisashi Shimodaira. A diversity-control-oriented genetic algorithm (DCGA): Performance improvement by the reinitialization of the population. In Lee Spector et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 576–583, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.
- [20] Tina Yu. Structure abstraction and genetic programming. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 652–659, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.