Controlling Exploration, Diversity and Escaping Local Optima in GP: Adapting Weights of Training Sets to Model Resource Consumption.

Tommaso F. Bersano-Begey
The University of Michigan Artificial Intelligence Laboratory
1101 Beal Avenue Ann Arbor, Michigan 48109-2106
tombb@engin.umich.edu

ABSTRACT

A common problem in evolutionary computation, and in particular in GA and GP, is the loss of diversity due both to 'lock-in' and early convergence of a population to 'deceptive' high scoring partial solutions. As the run proceeds, such individuals constitute a larger and larger segment of the population, eliminate diversity and stop the progress of the run. This paper will present a method for the automatic detection of such lock-in and impasse, as well as explore a fitness function which uses this information to reward diversity and innovation and increasingly penalize the locked-in individuals. This method can be applied to essentially any problem, and can be successful in preventing a run from locking-in on an easily obtainable local optimum and in preserving diversity. We present an analysis, discussion, and preliminary results on its application to the boolean 11-multiplexer problem.

1. Introduction

In [Langdon 96], Langdon analyzes in detail the evolution of a GP run (applied to the stack problem) and finds that the run converges too early and is also plagued by the presence of 'deceptive' high scoring partial solutions which cause a negative correlation between necessary primitives and fitness. He also noted that in later stages of the runs cross-over made increasingly less improvement, often duplicating individuals and thus causing a loss of variety while increasing convergence.

He then states that these problems are to some extent fundamental to GP, and proposes 2 possible improvements to a GP to partially address them: disabling the production of clones by the reproduction operator (to increase variety), and detecting when an offspring is identical to one of its parents (for the same purpose). Thus he seems to concentrate mainly on reducing duplicates. While such approach seems a reasonable improvement, this paper will concentrate on a more sophisticate technique which essentially makes use of additional information from the run's history and the individuals' phenotypes to detect early convergence to deceiving partial solutions and promote diversity directly through fitness.

The rest of this paper is organized as follows. In Section 2, I give an example of the problem caused by a certain type of lock-in, and introduce the concept of an alpha individual and other relevant concepts useful for my discussion. In Section 3, I describe and formalize a few ways to automatically detect such lock-in and escaping the consequent impasse. Then, in Section 4, I describe my first experiments and present my preliminary results. Finally, in Section 6, I state my conclusions and in Section 7 I discuss future work.

2. Analysis of Problem and Related Concepts

This paper examines the problem of lock-in of specific local optima solutions and proposes an automated method for detecting it and escaping from the impasse and loss of diversity that they create.

2.1 Alpha individuals

For ease of discussion, I use the arbitrary term of 'alpha individuals' to denote individuals which posses the following qualities:

1. They are combinatorially simple (e.g., depth 1-2 and equivalent ones at greater depths) and are thus present in large amounts since the early stages of a run.
2. They have a fairly high score, which is likely to be a higher score than that of any more complex individual in the first generations.
3. They are not useful genetic material (e.g., they are not building blocks for a perfect solution or a few mutations away from it).

2.2 A Sample Problem

To give a simple example of the dynamic of such individuals and their role in early convergence and lock-in, consider the following GP problem: draw the logic gate circuit for a boolean function which has 128 possible input combinations, 100 of which should output a 1, while the remaining 28 should output a 0.

Assuming that the terminal set contains also the value 0 and 1 (ground and vcc) or that these can be easily produced by one or more simple combinations of the terminals and functions (e.g., (AND x (NOT x))) it is clear that these are

an example of alpha individuals. In fact, these will have a very high score (100/128) which is likely to be one of the highest scores in the early generations. Thus, this might cause it to be selected over the rest of the population for most cross-over, mutation and reproduction operations. This can cause the run to lose much of its initial diversity and converge to such individuals, which, being alpha individuals, will not likely lead to a solution through cross-over or mutation. Also, since the other individuals in the population are not likely to score as high, they would be essentially left out and would have to start over at each generation, so that they would not be much better than random in producing competing individuals to challenge the alpha individuals.

In such cases, a run will typically degenerate by producing an increasing number of copies of such alpha individuals (similarly to [Langdon 96]), reducing diversity and effectively flattening the best-individual fitness curve for the remaining generations (as seen in Section 4).

2.3 Other approaches

An important issue discussed in this paper is: how can we recognize and counteract the tendency of a population to lock-in to an alpha individual and converge to it?

As mentioned before, [Langdon 96] suggests to concentrate on removal of duplicates to preserve diversity. Other related previous approaches are also focused on preserving diversity, for instance by using multiple pools of populations in parallel, allowing individuals to migrate between them to preserve diversity. Also, a large population might be able to find a better scoring non-deceiving individual before a deceiving or alpha individual capitalizes the population.

But if the problem is such that all the population tend to converge too soon to the same alpha individuals, each pool might obtain the same dominating individual (as might happen in the example in the previous subsection) and migration would involve mostly alpha individuals (thus losing its effectiveness). Also, given a deceptive function, any single mutation might not survive enough to re-establish diversity.

[Ryan 94] shows that maintaining increased diversity in GP leads to better performance.

Another very interesting analysis of run dynamics with respect to population diversity is presented in [Rosca 95], which suggests that GP diversity can be correlated with other statistical measures, such as average fitness. In it, Rosca uses a few experiments to look for a correlation between diversity and fitness change. However, it does not consider how the score is distributed between the fitness cases and thus the phenotype of the individuals, as this paper does, so that entropy measurements would not distinguish between different phenotypes which produce the same score.

Yet another approaches which also attempt to preserve diversity is found in [Eshelman & Shaffer 1993], which does not allow mating if two individuals are too similar (their Hamming distance is below a certain threshold).

3. Modified Fitness Function

Both the above analysis and [Langdon 96] would seem to indicate that such alpha individuals are responsible for early convergence and loss in diversity which often prevents a run from moving forward.

My solution to this problem is to recognize and counteract the tendency of a population to lock-in and converge to an alpha individual.

The basic idea is the following: to keep track of which fitness cases are solved by how many individuals in the population (H). In this way, it is possible to detect when the population is locked-in on a partial solution, by observing that at the phenotype level there are fitness cases which have been solved by too few individuals (H_i<c) in the past generations (W_g). Then, we can add an additional bonus to each fitness case which is a weight inversely proportional to how many other individuals in a population have already solved it. We can further increase this weight by counting how many generations this situation persists (W_g), so that we can put an increasing amount of evolutionary pressure over these fitness cases so as to gradually escape a lock-in, since the previously capitalizing individuals will be scoring increasingly less than newer individuals which are now solving the remaining fitness cases.

3.1 Resource Consumption

In the biological and evolutionary metaphor, this can be compared to a population of individuals feeding over a large field. As more and more individuals concentrate on small parts of the field, these areas will have less and less food, while the remaining areas will become more and more appealing in comparison.

In terms of ecological niches, we can imagine a similar situation in which there are different sources of food available, and as one is exploited more than another, there is an additional pressure to mutate and fill in another niche in which food is still available in large quantities.

Also, since we already store the best individual encountered so far, one can afford to penalize new copies of such individual.

This has some similarities with simple backtracking in search, although the previously wrong solutions are not stored and could potentially be repeated.

In terms of the algorithm, the new fitness function simply keep an array as large as the number of fitness cases, and as it evaluates each individual, it increases a counter for each of the respective fitness cases, so that after all the individuals have been evaluated, it would have stored also the relative percentage of individuals which solved each fitness case. We can then decide to store this information and either keep it around for the entire run (to get a complete history of the run) or replace it after each generation (to save memory, although the space should be negligible since it will take G*|int|*F bits, where G is #generations and F is #fitness cases).

Under such rules, an alpha individual would score increasingly lower as selection produced more copies of it and as it persisted through the generation, until eventually it might score lower than another individual which solves a set of fitness cases which were left relatively untouched by the rest of the population or in the past generations, ideally promoting the creation of phenotypic building blocks often complementary to the current most frequent phenotype.

With low epistasis this technique would likely work well, especially in producing building blocks which could then be recombined through cross-over. However, while genotypic building blocks could potentially recombine nicely, phenotypic building block recombination might not necessarily ever lead to an offspring with the phenotype of both parents combined.

Also, high epistasis will complicate things further, because it becomes then unclear if the interaction between fitness cases will make the change of weight strategy ineffective.

For these reasons it is important to refer back to the original metaphor of evolution and natural selection in biological entities, to see if this technique can be meaningfully mapped into an existing phenomenon, thus reinforcing the hypothesis that a similar method could be successful in those situations regardless of all the above problems.

3.2 New fitness functions in detail

In slightly more mathematical terms and in spelling out the new fitness function, we can write:

Legend:

fc = number of fitness cases,

n = number of individuals,

Dt = runtime parameter: threshold number of generations for detecting an impasse.

Cg = counter, increases for each generation in which the best individual's fitness is the same as the previous generation's. Resets to 0 otherwise.

$$F_{old}(IND,i)=\begin{cases}1 & \text{if the individual } IND \text{ solves the } i\text{th fitness case}\\ 0 & \text{otherwise}\end{cases} \qquad -Eq1-$$

$$F_{old}(IND_k)=\sum_{i=1}^{fc}F_{old}(IND_k,i) \qquad -Eq2-$$

Equations 1 and 2 show the normal fitness function for counting hits or raw fitness in GP: *Fold* is essentially the number of fitness cases successfully solved by an individual (*INDk*).

$$H_i=1+\sum_{k=1}^{n}F_{old}(IND_k,i) \qquad -Eq3-$$

$$W_g=\begin{cases}0 & C_g\le D_t\\ C_g-D_t & C_g>D_t\end{cases} \qquad -Eq4-$$

$$F_{new1}(IND_k)=\sum_{i=1}^{fc}\frac{F_{old}(IND_k,i)}{(H_i)} \qquad -Eq5-$$

$$F_{new2}(IND_k)=\sum_{i=1}^{fc}\frac{\frac{W_g}{(H_i)}+F_{old}(IND_k,i)}{1+W_g} \qquad -Eq6-$$

$$F_{new3}(IND_k)=\sum_{i=1}^{fc}\frac{F_{old}(IND_k,i)}{(H_i*W_g+1)} \qquad -Eq7-$$

Equations 3 to 7 show a variant which is an immediate translation of the metaphor and method described in the previous section.

Equation 3 describes the array which stores the hits distribution of all the *n* individuals over any one *i* fitness case, while Eq.4 describes W_g, which becomes non-zero when an impasse has been detected, increasing as the impasse persists over the generations.

Equation 5 is essentially introducing a weight connected to each fitness case which is inversely proportional to the number of individuals which also solve it (H_i).

In Eq.6, the new score is produced by adding the regular score (e.g., eq.2) a bonus proportional to both the number of generations the impasse persisted (W_g) and how many other individuals also solve the same fitness case (H_i). I then normalize the score (by dividing by $1+W_g$). Note that when W_g=0 (when there is no impasse), Eq.6 becomes equivalent to Eq.2. Equation 7is another variant of Eq.6, but it behaves in a similar matter.

After trying a few runs, however, I obtained discouraging preliminary results with the three fitness functions of equations 5, 6, and 7, probably because they excessively inflated the fitness value of individuals, so that while they were successful in allowing the run to escape an impasse, the altered weights did not reflect the actual value of the individuals.

For this reason, I slightly altered the previous equations to ensure that the weights would reflect actual performance (number of hits), except for the detected alpha individuals during an impasse, and arrived at equations 8,9, and 10.

$$B_{k,\,i}=\begin{cases}1 & F_{old}(IND_{best},i)=0\wedge F_{old}(IND_k,i)=1\\ 0 & \text{otherwise}\end{cases} \qquad -Eq8-$$

$$B_k=\sum_{i=1}^{fc}B_{k,\,i} \qquad -Eq9-$$

$$F_{new4}(IND_k)=\begin{cases}F_{old}(IND_k) & (W_g=0)\vee(B_k>0)\\ \frac{F_{old}(IND_k)}{W_g} & (W_g>0)\wedge(B_k=0)\end{cases} \qquad -Eq10-$$

Equation 8, 9, and 10 essentially assign *Fold* to all the individuals, except for the best scoring alpha individual during an impasse, whose score is weighed by the inverse of W_g. Also note that I no longer compute H_i, but rather use only the distribution of hits of the best scoring alpha individual.

4. Preliminary Results

4.1 Methodology

Runs were executed on a Sun Ultra using a modified code from the lilgp[ref 1] GP kernel and examples, and applied to solve the 11-Multiplexer problem described by Koza [Koza 92]. I chose that problem mainly because it has the tendency of creating alpha individuals and impasses. In fact, note that this problem is essentially a larger and more balanced version of the sample problem described in Section 2. In the experiment I compare a the two fitness functions (Eq2 and Eq4) by analyzing the dynamics of each run's fitness curve.

4.2 Run parameters

I used 3 sets of runs. The first set of runs consisted of 120 runs using 50 generations and a population size of 20. Half of the runs used the standard fitness function of Eq.2, and the remaining half used the modified fitness function of Eq.10. I used random seeds from 1 to 60 for each half respectively.

I chose a particularly small population size in Run1 mainly because I wanted to see the effects of my function when the run cannot rely much on a large population to preserve some diversity and find a solution better than a dominant alpha individual.

The second set of runs consisted of 5 runs using 1000 individuals and 50 generations, and the third set of runs consisted of 5 runs using 2000 individuals and 50 generations.

4.3 Discussion of the results

I have not yet graphed the statistical results summarizing the entire set of runs, but in this experiment I

believe that the dynamic of each single run (and in particular of the best score in each generation) is more significant.

However, to summarize the results of the sets of runs, in all three sets the modified fitness function always succeeded in detecting and escaping impasse (although while impasses were the norm in the set 1, they become less frequent in the other sets).

The first set also obtained always equal or better results (higher score of the best-of-run individual), while the other two sets often obtained better solutions with the standard fitness function. One possible reason for this might be that the impasse was created by non-alpha individuals, which, when maintained in through generation, eventually produce a better scoring individual through mutation or crossover.

In examining the dynamic of individual runs I selected out a few runs from the first set which are somewhat representative.

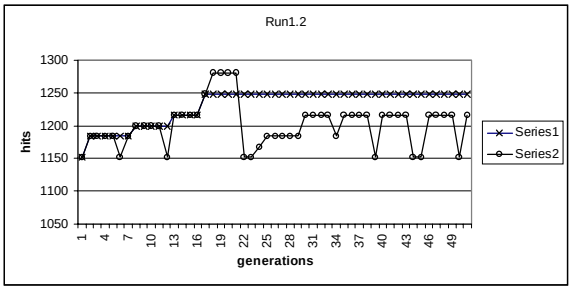

Figure 1. is an example of a successful use of the modified fitness function. Series 1 represents the standard fitness function, which grows until generation 16 and then remains stuck on the same best individual for the remaining 34 generations. The modified function, instead, detects impasses at generation 6 and 12, and is later able to find a higher scoring solution (generations 18-20).

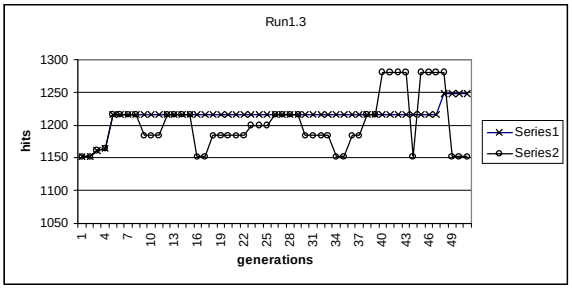

Figure 2. is another example of a success of the modified fitness function. In this case we see that the normal function finally escapes the impasse after about 40 generations, but the modified function kept on exploring obtaining ups and downs in the fitness curve, and eventually obtained a much better scoring solution.

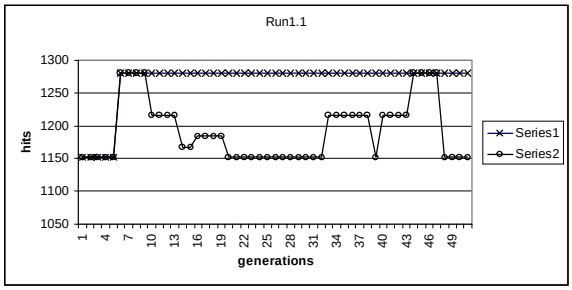

Figure 3. is an example of a case in which the modified fitness function does only as good as the normal one

5. Conclusions and Future Work

I have presented a method for automatic detection of early convergence of a population to a local optima, proposed an account for loss of diversity in certain GP problems, and proposed different approaches which take advantage of additional information on the phenotype, entropy and fitness distribution of a population during a GP run. Preliminary runs seem to suggest that a modified fitness function based on the principle of resource consumption can be successful in escaping lock-in of partial solutions and the impasse or plateau of best-of-generation plots which seems to accompany them. In some instances (such as low population runs), this often results in more fit best-of-run individuals.

However, because of the little experimental evidence in this preliminary work, the results are not yet conclusive on the effectiveness of this method. In future work I will continue these experiments to measure population diversity through population fitness histograms, get a more detailed statistical picture of the runs (comparing normal and modified fitness functions) and examining best-of-generation individuals during an impasse to verify the alpha individual hypothesis.

The general method found in this paper can be applied to essentially any problem, although it is most useful for those problems which suffer from loss of diversity and which might contain alpha individuals. Furthermore, the relevance of this method relies not only on its ability to escape an impasse and promote diversity, but also in its potential to direct exploration of different and possibly complementary phenotypes (which could be useful for creating matching building-blocks).

Acknowledgments

I Thank Jason Daida for its many contributions and insightful discussions, E. Durfee, R.Bertram, C. Grasso, J. Polito, S.Stanhope and A&M Bersano-Begey for their support.

Bibliography

[ref 1] http://isl.cps.msu.edu/GA/software/lil-gp/

[Eshelman & Shaffer 93] Eshelman, L.J., Shaffer, J.D. 1993. *Crossover's niche*. In Forrest, S. (editor) Proceedings of the International Conference on Genetic Algorithms, pages 9-14. Morgan Kaufmann.

[Koza 92] Koza, John. R. 1992 *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

[Langdon 96] W. B. Langdon. 1996d. *Evolution of Genetic Programming Populations*. University College London technical report RN/96/125. September 1996.

[[Rosca 95] J.P. Rosca. 1995. *Entropy-Driven Adaptive Representation*. In J.P.Rosca (editor) Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications. Tahoe City, CA. Pages 23-32.

[Ryan 94] C.O. Ryan. 1994. *Pygmies and civil servants*. In Kinnear, K., editor, Advances in Genetic Programming. MIT Press.