Why "Building Blocks" Don't Work on Parity Problems

W. B. Langdon and R. Poli
School of Computer Science
The University of Birmingham,
Birmingham B15 2TT, UK

{W.B.Langdon,R.Poli}@cs.bham.ac.uk
http://www.cs.bham.ac.uk/~wbl, ~rmp
Tel: +44 (0) 121 414 4791 Fax: +44 (0) 121 414 4281

Technical Report: CSRP-98-17

13 July 1998

Abstract

We investigate the distribution of performance of the parity and always-on Boolean functions given only the appropriate building block. These problems have "needle in a haystack" fitness landscape and so are unsuitable for genetic programming or other progressive search techniques. Theoretical analysis shows in the limit as program size grows the density of solutions is independent of size but falls exponentially with number of inputs.

1 Introduction

The Boolean parity functions have been over used as benchmark problems in GP. Investigations are usually empirical and often contain little analysis. The benchmarks are considered difficult, with the standard function set only the smaller versions of the problem can be solved by random search [Langdon and Poli, 1998a]. Perhaps their difficulty arises because the standard representation does not allow the ready discovery and use of building blocks? A number of techniques (ADF [Koza, 1994], ARL [Rosca, 1997], GLiB [Angeline, 1993]) have been suggested that aim to ease the discovery and use of building blocks. We assume some such approach has been successful at evolving functional building blocks and analyse the fitness space given a function set which comprises only this building block.

In Section 2 we describe the Boolean functions and give an expression for the number of them which can be constructed with one binary function. The form of the solutions and the building block used are given in Section 3 and in Section 4 we analyse the problem and derive a limit for the density of solutions. This is followed, in Section 5, by a discussion of the significance of this result and by our conclusions (Section 6).

2 Size of the Search Space

The program trees we will consider are composed of n terminals (D0, D1, ... D_{n-1}) which are the Boolean inputs to the program and the Boolean logic functions EQ or XOR [Koza, 1992]. (D0, D1, ... D_{n-1}) and EQ are sufficient to create an even-n parity function, when n is even and likewise (D0, D1, ... D_{n-1}) and XOR are sufficient to create an odd-n parity function if n is odd. (Odd parity solutions can be readily converted to even parity by inverting the final output of the program. For simplicity we don't require the final negation and instead consider programs with only one type of function). The fitness of each tree is given by evaluating it as a logical expression for each of the 2^n possible combinations of D_n inputs. Its fitness is the number of fitness cases when its output agrees with that of the target Boolean function.

There are $n^{(l+1)/2} \times \frac{(l-1)!}{((l+1)/2)!((l-1)/2)!}$ different trees of length l [Koza, 1992, page 213] [Alonso and Schott, 1995]. The number of programs rises rapidly (approximately exponentially) with increasing program length l. Of course if no bounds are placed on the size or depth of programs then the number of them is unbounded, i.e. the search space is infinite.

3 Parity and Always-on/off Program Spaces

Even parity solutions, where n is even, are of the form $D0 = D1 = D2 = \ \ldots\ = D_{n-1}$. However given the symmetry of the EQ building block the inputs to the program can occur in any order. Further $D_x = D_x$ is true so $(D_y = D_x = D_x) = D_y$ therefore any pair of repeated inputs can be removed from the program without changing its output.

Odd parity solutions, where n is odd, are of the form $D0 \neq D1 \neq D2 \neq \ \ldots\ \neq D_{n-1}$. Again given the symmetry of the XOR building block the inputs to the program can occur in any order. Further $D_x \neq D_x$ is false but $(D_x \neq \text{false}) = D_x$ so $(D_y \neq D_x \neq D_x) = D_y$ therefore again any pair of repeated inputs can be removed from the program without changing its output.

Therefore any program will be a solution to the parity problem provided it contains an odd number of all n terminals. Thus all solutions contain $t = n + 2i$, where $i = 0, 1, 2, \ldots$, terminals. Both EQ and XOR are binary functions, so programs are of odd length $l = 2t - 1$ and solutions are of length $l = 2t - 1 = 2n + 4i - 1$.

Any program with an even number of one or more inputs effectively ignores those inputs. Given the nature of the parity function, such a program will pass exactly half the fitness cases.

The always-on function effectively ignores all its inputs. When n is even this can only be implemented by having an even number of all inputs. If there are an odd number of any input, then the function will return true in exactly half the fitness cases. If n is odd, always-on cannot be implemented using the available primitives (XOR and D_i) however the analogous function always-off can and like always-on (for the n even case) requires there to be an even number of each input. Otherwise it returns false in exactly half the fitness cases.

4 Proportion of Solutions

If a program's length does not obey $l = 2n + 4i - 1$ then it cannot be a solution to the order n parity problem and will score exactly half marks on the parity problems. If $l = 2n + 4i - 1$ is true there is a chance that a randomly generated program will contain an odd number of each input and so be a solution.

When calculating the fraction of programs of a given length that are solutions we can ignore the number of different tree shapes because whether a tree is a solution or not is determined only by how we label its leafs and not by its shape.

The proportion of trees of length l which are parity solutions is the same as the chance of choosing at random t coloured balls, with reselection, from a bag containing equal numbers of each n-coloured balls, in such away that we have an odd number of all n-colours. (The proportion which are solutions to the always-on or always-off problems are the chance that we have an even number of all n-colours). The probability of drawing b_1 balls of the first colour, b_2 of the second colour and so on to b_n balls of the last colour, is given by a multinomial distribution

$$P(b_1, b_2, b_3, \ldots\ b_n) = \frac{t!}{b_1! b_2! b_3!\ \ldots\ b_n!} 1/n^t \tag{1}$$

While drawing each ball is an independent event, the chance of having an odd number of one colour is not independent of the chance that we have an odd number of a different colour. This is because we are drawing exactly t balls and so the sum of each colour must come to t. Calculating these probabilities and summing them is tedious but we can derived a limit for behaviour as the number of balls becomes large.

Consider first one colour (one terminal, e.g. D0). The chance of drawing an odd number of them from the bag in t trials is $\sum_{j=1,3,5,\ldots}^{t} C_j^t \frac{(n-1)^j}{n^t}$ (where $C_j^t = \frac{t!}{j!(t-j)!}$). I.e. the sum of all the odd terms in a binomial distribution. As the number of trials in a binomial distribution is increased it becomes increasingly smooth. (In the limit it approaches the Gaussian distribution, with a central peak at t/n and a standard deviation of $\sqrt{t(n-1)}/n$). If $\sqrt{t/n} \gg 1$ then each term in the binomial expansion is close to its neighbour, i.e. each odd term is close to an even term so $\sum_{j \in \mathrm{odd}}^{t} C_j^t \frac{(n-1)^j}{n^t} \approx \sum_{j \in \mathrm{even}}^{t} C_j^t \frac{(n-1)^j}{n^t}$ and since $\sum_{j=0,1,\ldots}^{t} C_j^t \frac{(n-1)^j}{n^t} = 1$ so $\sum_{j \in \mathrm{odd}}^{t} C_j^t \frac{(n-1)^j}{n^t} \approx \frac{1}{2}$ and $\sum_{j \in \mathrm{even}}^{t} C_j^t \frac{(n-1)^j}{n^t} \approx \frac{1}{2}$.

The total number of each of the colours must sum to t. This reduces the number of degrees of freedom by one, i.e. given the oddness of the number of balls of $n-1$ colours the oddness of the number of the remaining colour is fixed. Thus the total probability that there will be an odd number of each colour is the product of $(n-1)$ rather than n independent events, i.e. $1/2^{n-1}$. Likewise the probability of an even number of every colour will also tend to $1/2^{n-1}$ as t increases. ([Fenner, 1998] has a formal proof of this result using a Markov chain analysis).

Figure 1 shows the fraction of 100,000 random programs which are solutions to the Even-6 parity or always-on-6 problems, for a variety of lengths. Figure 1 shows measurement approaches the theoretical large program limit closely for $t \geq 16$, i.e. $l \geq 31$.

Figure 2 shows the fraction of random programs which are solutions to the Even or Odd parity problems (100,000 or a million random trials). The lengths of the programs were chosen so that $l = 2n + 4i - 1$ is true and the number of leafs exceeds $6/16 \times n$. Figure 2 again shows agreement between measurement and the theoretical large program limit.

5 Discussion

Section 4 shows the program space of the parity functions when given the appropriate functional building block. For all but the simpliest problems at all program lengths the fitness space is dominated by a central spike indicating almost all programs score exactly half marks. However a small fraction of programs do solve the parity problems. We have derived a simple analytical expression for the case of large programs which shows, the proportion of solutions falls exponentially with increasing number of inputs. However for modest, but realistic, numbers of inputs the proportion is not so small as to be infeasible to find using modern computers.

To derive a fitness landscape, we would also need to consider how genetic or other operators move between points in the search space, as well as the fitness of those points. However this is unneccessary in this case because either we have found a solution or the point in the search

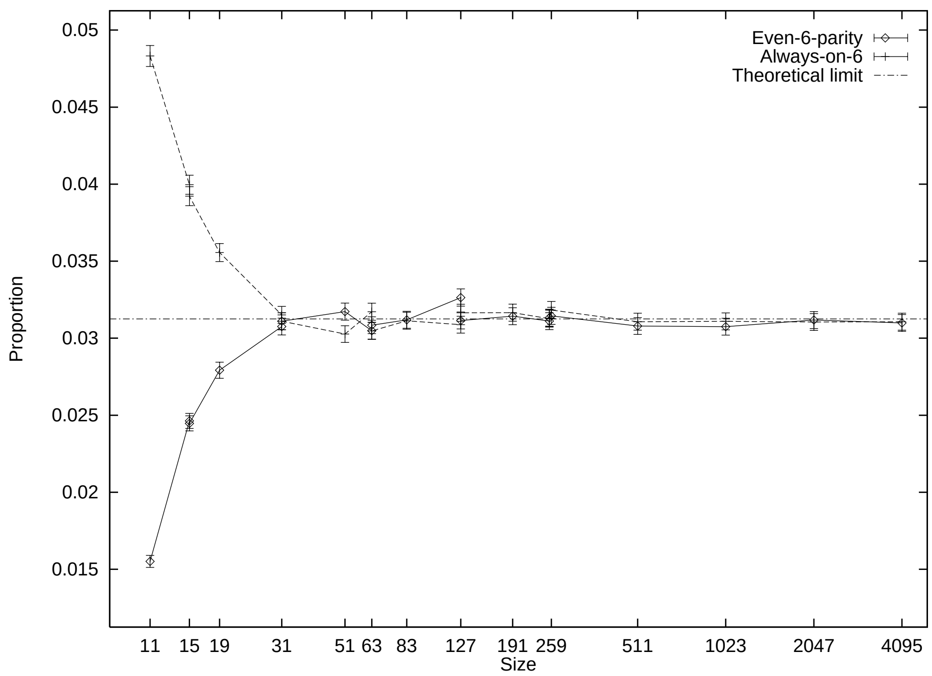

Figure 1: Fraction of programs that solve Even-6 parity or Always-on-6 problems, using EQ (note log scale). Error bars show standard error.

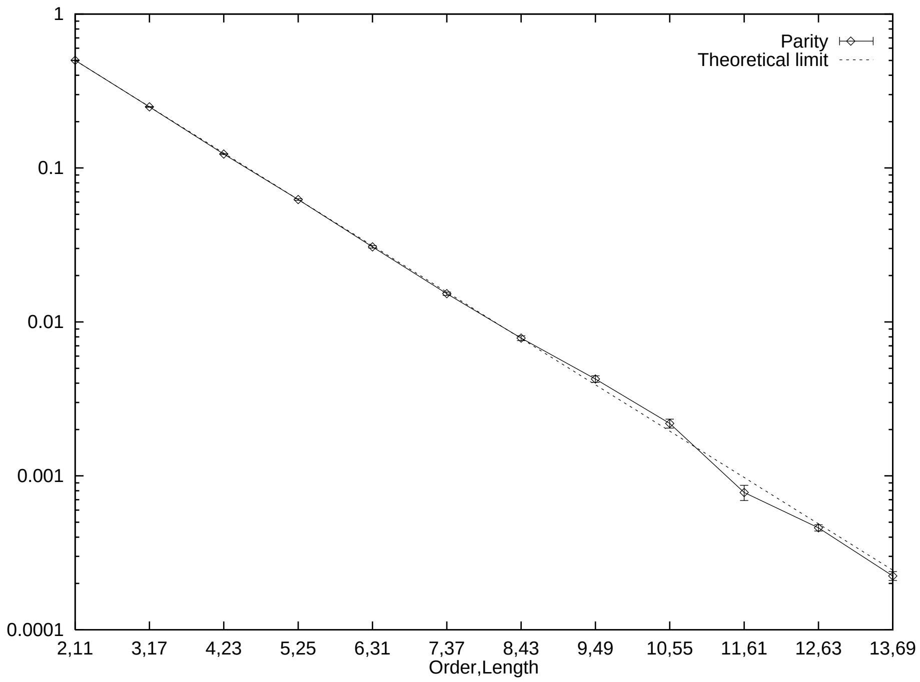

Figure 2: Fraction of programs that solve parity problems given appropriate building block. Length of programs increases with problem order so leafs $\geq 2\frac{2}{3}$order.

space scores half marks. That is the program space contains no gradient information. Search techniques, such as GP, which rely on gradient information will be unable to out-perform random search on such a landscape. Indeed we might anticipate population based search without mutation performing worse than random search as genetic drift in a small population means the population may loose one or more primitives. If this happended then it becomes impossible to construct a solution.

This suggests techniques which seak to solve the parity problems by evolving the appropriate building blocks are unlikely to find minimal solutions directly. Such evolutionary technqiues will probably have found programs with fitness well above half marks [Langdon and Poli, 1998a] and so will reject almost any solution only composed of the discovered building blocks since most of these will score less than the partial solutions it has already discovered. (It is possible impure solutions to the problem may be found which subsequent evolution, perhaps under the influence of parsimony or beauty pressures, may convert into a solution comprised only of building blocks).

When considering the Boolean problems it is common to require the inputs to be unstructured (a notable example gainfully employing typed inputs is [Janikow, 1996], while Yu structures her inputs as a list [Yu and Clark, 1998]). Our analysis shows it may not be enough to find functional building blocks without considering how they will act on the inputs.

If we compare Figure 1, with a similar plots for the artificial ant problem on the Santa Fe trail [Langdon and Poli, 1998b, Figure 2] and plots for Boolean problems with other function sets [Langdon and Poli, 1998a] we see the proportion of programs which solve a particular problem or reach a given level of performance on the problem converges to a stable value which appears to be more-or-less independent of program size (provided we ignore lengths which syntactically cannot be solutions). We have demonstrated this property on the 256 3-input Boolean functions, 6 6-input Boolean functions as well as the ant problem. We expect this property to hold in many cases, however demonstrating it on 263 problems is not sufficient to prove it holds in general. However in Section 4 we proved it will hold in general for the special cases of the parity and always-on problems when given appropraite building blocks. These add weight to some of our claims about the nature of program fitness landscapes and their influence on the bloating phenomena [Langdon and Poli, 1997].

Random search using the ramped-half-and-half method of creating random trees [Koza, 1992, page 93] is poor at finding solutions to these parity and always-on problems because it creates trees of many sizes. Most of these do not obey the restriction $l = 2n + 4i - 1$ and so will score exactly half marks on these problems.

6 Conclusions

Even when given appropriate building blocks the Boolean parity problems are still difficult. The fitness landscapes are a "needles-in-haystacks" with the chance of random search finding a solution falling exponentially with number of inputs. We conclude solving such problems efficiently also requires consideration of how the inputs are structured. Work is continuing to understand the role of different function sets and search operators on the Boolean problems.

In two very different classes of problems (the Ant and Boolean problems) we have now shown that the fitness space is in a gross manner independent of program length. In general the number of programs of a given length grows approximately exponentially with that length. Thus the number of programs with a particular fitness score or level of performance also grows exponentially, in particular the number of solutions also grows exponentially.

Acknowledgements

This research was funded by the Defence Research Agency in Malvern.

References

[Alonso and Schott, 1995] Laurent Alonso and Rene Schott. *Random Generation of Trees.* Kluwer Academic Publishers, 1995.

[Angeline, 1993] Peter John Angeline. *Evolutionary Algorithms and Emergent Intelligence.* PhD thesis, Ohio State University, 1993.

[Fenner, 1998] Trevor I Fenner. Limiting distributions of odd coloured balls. Personal Communication, July 1998.

[Janikow, 1996] Cezary Z. Janikow. A methodology for processing problem constraints in genetic programming. *Computers and Mathematics with Applications*, 32(8):97–113, 1996.

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

[Koza, 1994] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge Massachusetts, May 1994.

[Langdon and Poli, 1997] W. B. Langdon and R. Poli. Fitness causes bloat. In P. K. Chawdhry, R. Roy, and R. K. Pan, editors, *Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing.* Springer-Verlag London, 23-27 June 1997.

[Langdon and Poli, 1998a] W. B. Langdon and R. Poli. Boolean functions fitness spaces. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Stanford University Bookstore.

[Langdon and Poli, 1998b] W. B. Langdon and R. Poli. Why ants are hard. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 193–201, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[Rosca, 1997] Justinian P. Rosca. *Hierarchical Learning with Procedural Abstraction Mechanisms.* PhD thesis, University of Rochester, Rochester, NY 14627, February 1997.

[Yu and Clark, 1998] Tina Yu and Chris Clark. Recursion, lambda-abstractions and genetic programming. In Riccardo Poli, W. B. Langdon, Marc Schoenauer, Terry Fogarty, and Wolfgang Banzhaf, editors, *Late Breaking Papers at EuroGP'98: the First European Workshop on Genetic Programming*, pages 26–30, Paris, France, 14-15 April 1998. CSRP-98-10, The University of Birmingham, UK.