Boolean Functions Fitness Spaces

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-98-16
15 June 1997 (22 August 2002)*

ABSTRACT

We investigate the distribution of performance of the Boolean functions of 3 Boolean inputs (particularly that of the parity functions), the always-on-6 and even-6 parity functions. We us enumeration, uniform Monte-Carlo random sampling and sampling random full trees. As expected XOR dramatically changes the fitness distributions. In all cases once some minimum size threshold has been exceeded, the distribution of performance is approximately independent of program length. However the distribution of the performance of full trees is different from that of asymmetric trees and varies with tree depth.

We consider but reject testing the No Free Lunch (NFL) theorems on these functions.

1 Introduction

Our investigations of the artificial ant following the Santa Fe trail [Langdon and Poli, 1998] suggests that, provided programs are big enough, the distribution of program fitnesses is roughly independent of their size. That is if we pick a program of a certain length at random its as likely to perform as well as another program of a different length also chosen at random (provided both exceed some threshold size). If this is generally true then as the size of the search space grows approximately exponentially as we allow longer programs then so to does the number of programs with a certain level of performance. Therefore while a search space with no size or depth limits will be infinite it will contain an infinite number of solutions!

We test this result from the Ant problem on a range of other problems. In Section 2 we describe the Boolean problems. Section 3 describes how we measure the performance spaces of these problems and gives our results. The ramped-half-and-half method [Koza, 1992, page 93] is commonly used to generate the initial population in genetic programming (GP). Half the random programs generated by it are full. Therefore we also explicitly consider the subspace of full trees. This is followed by a discussion of these results and their implications (Section 4) and our conclusions (Section 5). Finally in Appendix A we consider using the Boolean functions to experimentally test the No Free Lunch theorems and give reasons to reject this idea.

2 Boolean Functions

The Boolean functions have often been used as benchmark problems. 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 AND, OR, NAND and NOR [Koza, 1992]. These are sufficient to construct any Boolean function but we shall also investigate including the exclusive-or function (XOR). Note [Koza, 1992] required all the functions to have the same arity, this is not required in our approach. 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 $n^{(l+1)/2}|F|^{(l-1)/2} \times \frac{(l-1)!}{((l+1)/2)!((l-1)/2)!}$ different trees of length l [Koza, 1992, page 213] [Alonso and Schott, 1995]. $|F|$ is four (or five if XOR is included). (Note this formula is simple as each function (internal node) has two arguments). The number of programs rises rapidly (approximately exponentially) with increasing program length l (see Figures 1 and 2). 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.

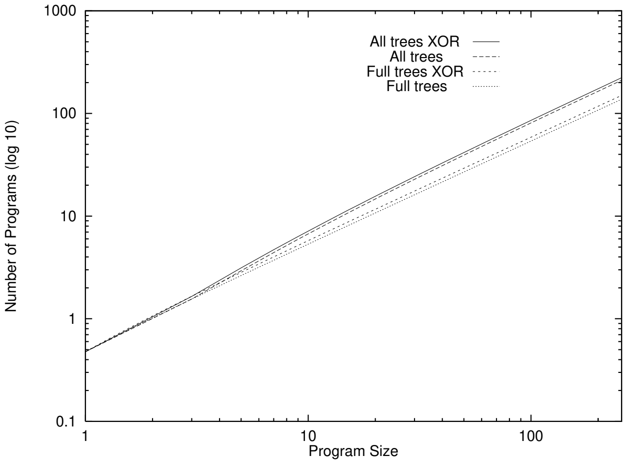

Figure 1 Size of 3-Input Boolean Function Search Spaces (note log log scale)

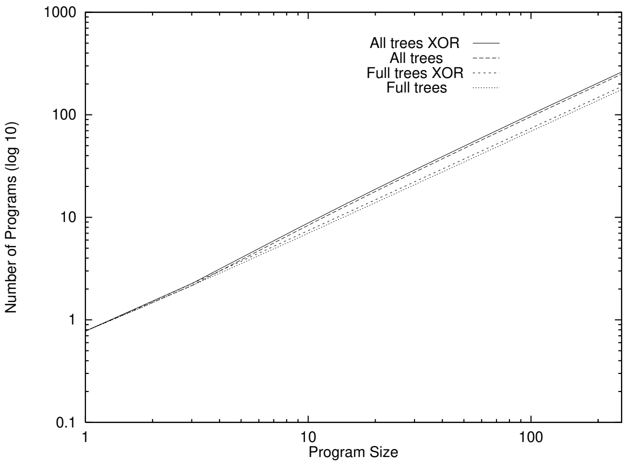

Figure 2 Size of 6-Input Boolean Function Search Spaces (note log log scale)

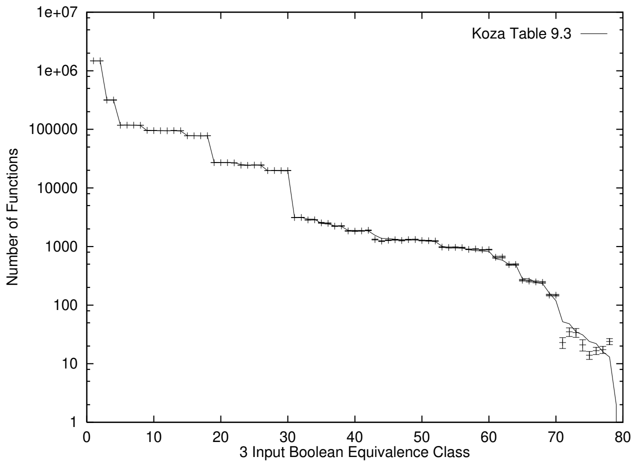

Figure 3 Number of functions of length 41 in each equivalence class

3 Boolean Program Spaces

3.1 3 Input Boolean Program Spaces

Recall that the fitness of a program is determined by the closeness of the function the program actually implements and the target function. E.g if the program implements function 0 (always return false) its fitness when searching for 3 input rule 150 (3-even parity) is 4, since it gets half of the 8 test cases right. That is for each target function there is a fixed simple mapping between the function implement by a program and the program's fitness. So the functionality of a trial solution readily gives its fitness on all the Boolean problems. Therefore by considering the distribution of the functionality of each point in the search space, we can consider simultaneously the fitness distribution of all of the Boolean functions.

In this section we consider all the Boolean functions for $n = 3$. There are 256 of them but they can be split into 80 equivalence classes. Functions are equivalent if permuting their inputs can produce the same functionality. By symmetry members of the same equivalence class will occur in the same numbers in the search space. Therefore we need only consider one representative function from each class [Koza, 1992, page 215].

Figure 3 shows the number of examples of each function found when 10 million trees of length 41 were created at random. (The 80 equivalence classes are ordered along the x-axis in order of decreasing frequency). As expected there is good agreement with [Koza, 1992, Table 9.3]. Figure 4 shows similar plots for a wide range of lengths. (Data for lengths 1, 3, 5, 7, 9, 11 and 13 were gathered by evaluating every tree of that length, whereas data for larger trees was gathered by randomly sampling 10 million programs for each length. C++ code to generate random programs is available via `http://www.cs.bham.ac.uk/~wbl`).

Figure 4 shows a certain minimum size is required before the problem can be solved and that the minimum size depends on the difficulty of the problem. Once this threshold size is exceeded the proportion of programs which belong to the equiv-

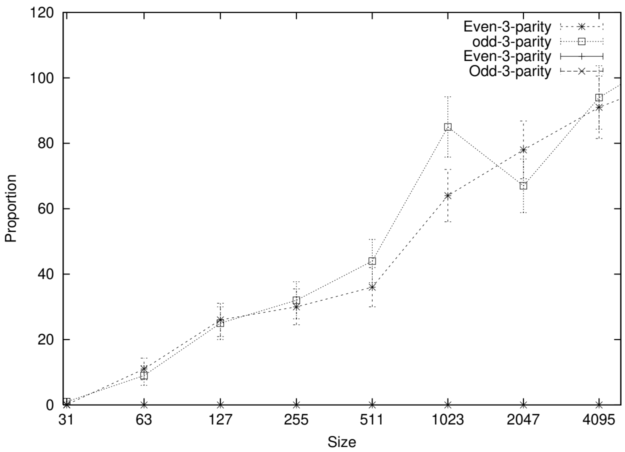

Figure 6 3 input parity functions in full trees

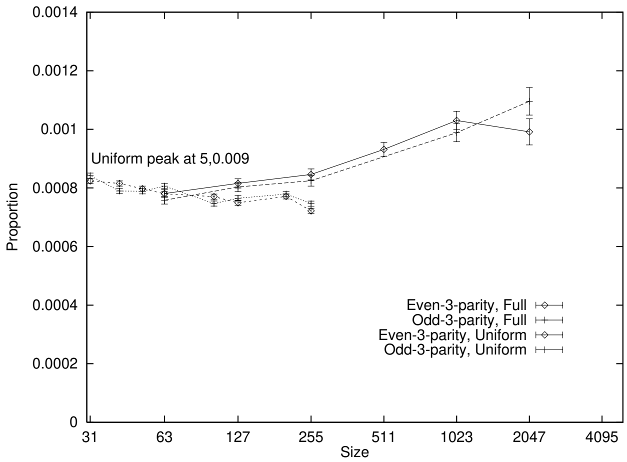

Figure 7 Proportion of 3 input parity functions in full trees with XOR

alence class grows rapidly to a stable value which appears to be more-or-less independent of program size. Figure 5 shows these characteristics are retained if we extend the function set to include XOR. (Adding XOR to the function set greatly extends the search space and so enumerating all trees of length 13 is no longer feasible, therefore data for length 13 was produced by random sampling). Note adding the XOR function radically changes the program space. In particular, as might be expected, the two parity functions (equivalence classes 79 and 80) are much more prevalent. Also the range of frequencies is much reduced. For example 68 of the 80 equivalence classes have frequencies between $0.1/256$ and $10/256$ rather than 28 with the standard function set.

While Figures 4 and 5 can be used to estimate the fitness space of each three input Boolean function across the whole space, there are some interesting parts of these spaces where certain functions are more concentrated than elsewhere. Figure 6 plots the proportion of full trees of different depths which implement the parity functions. It is clear there are far more parity functions amongst the full trees than there are on average. When XOR is added to the function set, see Figure 7, there are again a higher proportion of parity functions but the difference between the full trees and the rest of the search space is less dramatic.

3.2 6 Input Boolean Program Spaces

It is not possible to analyse all the Boolean functions with more than three inputs. Instead we have concentrated on what are generally considered to be the easiest and hardest Boolean functions of six inputs. Namely the always-on-6 function and the even-6 parity function. Figures 8 and 9 show the proportion of programs of various lengths with each of the possible scores. Figures 10 and 11 show the same when XOR is added to the function set. It is clear all four problems have the same near independence of fitness from length.

Figure 9 shows a huge peak with 90% of points in the search space having a fitness on the even-6 parity problem of exactly half marks, i.e. 32. The number of programs with other scores falls exponentially away either side of the peak. Even sampling 10,000,000 points per length, only three programs (1 with 27 and two with 37) were found outside the range $28 \ldots 36$ hits. [Rosca, 1997, Figure 4.1] reports similar behaviour on the even-5-parity problem. (Note that he used ramped-half-and-half, only sampled 16,000 programs and did not consider variation of the fitness distribution with length). He reports that the fitness distribution for the even-5-parity problem are even more tightly grouped in a range of 5 values. This could be due to the smaller study size but allowing for this, the comparable range for even-6 parity still spans 7 values. Looking at the short programs in Figure 9 shows they have an even tighter distribution of fitness. If this is also true for the even-5 parity problem then the range reported in [Rosca, 1997] will be due to the larger of the trees he sampled, particularly the full trees. It seems reasonable to suggest that the difference between a range of fitness values reported by [Rosca, 1997] on the even-5 parity problem (5) and that we find on the even-6 parity problem (7 or 9) is indeed due to the peak in the fitness distribution being wider, rather than an artifact of the bias inherent in ramped-half-and-half.

The fitness distribution of the even-6 parity problem is much tighter than that of the binomial distribution that would be produced by selecting Boolean functions uniformly at random from the 2^{2^n} available. I.e. centred on $\frac{n}{2}$ with variance of $\frac{n}{4}$ [Rosca, 1997, page 62]. The measured variance is only 0.12 rather than 1.5. Such a tight fitness distribution and in particular the absence of a high fitness tail suggests that the problem will be hard for any adaptive algorithm.

As expected adding XOR to the function set greatly increases the even-6 parity fitness distribution's width and it retains its near independence of program size (see Figure 11). The standard deviation is now 0.92 rather than 0.34. However the more dramatic effect of the wider distribution is the number of solutions (i.e. programs scoring 64 hits) is now measurable and is about $2\ 10^{-7}$.

Figure 8 shows the distribution of number of trues returned is a saw toothed curve. The proportion of programs which

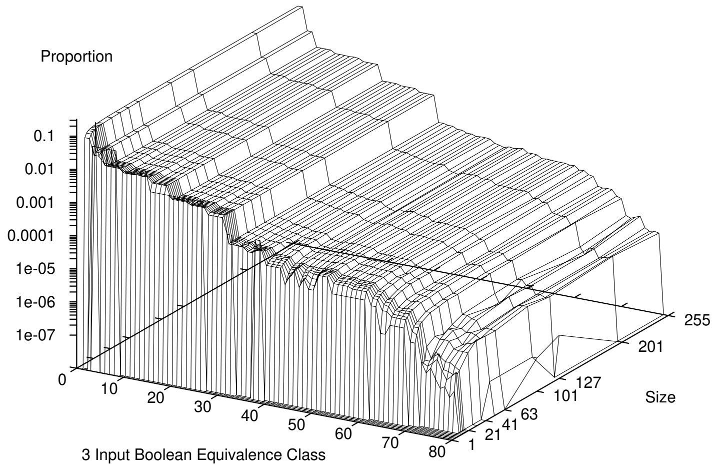

Figure 4 Proportion of functions in each equivalence class

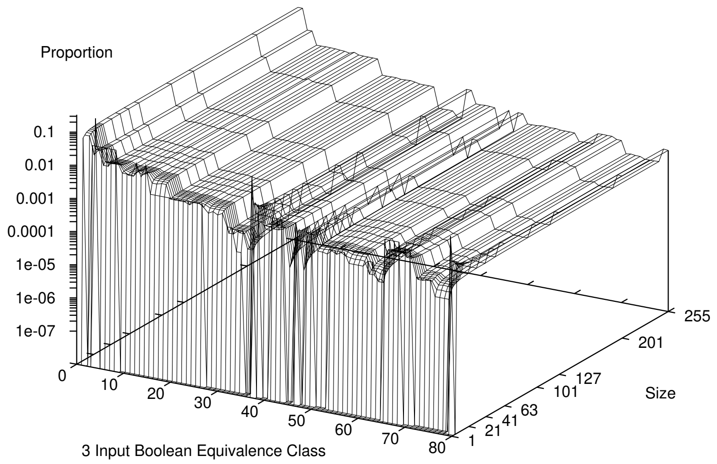

Figure 5 Proportion of functions in each equivalence class with XOR in function set

have one of the odd scores on the always-on-6 problem is about 0.3%. The proportion which have an even score, not divisible by four, is about 1%, scores divisible by 4 about 2%, those by 8 3%, those by 16 6% and those by 32 10%. Note the central peak in the even-6 parity fitness distribution (see Figure 9) is not solely due to a large number of programs which implement always-on-6 or always-off-6. Only 18.6% of programs are of these two types.

Figure 10 shows the distribution of number of trues returned when XOR is added to the function set is a little changed (cf. Figure 8) but retains its saw toothed appearance and near independence of program size.

3.3 Even-6 Parity and Always-On-6 Full Trees

Restricting our search to just the full trees yields a similar fitness distribution for the even-6 parity problem, see Figure 13. However the distribution of fitness values is considerably wider with a range of 25–38 (twice that for the whole search space) and a standard deviation of 0.68. Adding XOR to the function set (see Figure 15) further widens the distribution (the standard deviation becomes 1.8). Even when including XOR, solutions are so rare that their proportion is difficult to estimate accurately. Given this the proportion of solutions in the full trees appears to be the same as in the rest of the search space (with the same program length) despite an overall spreading of the fitness distribution. (The only even-6 parity programs found by random search through full trees contained either 15 or 31 nodes). Both with XOR and without the distribution of hits on the even-6 parity problem returned by full trees shows some dependence on depth of tree but this is much less dramatic than is the case with the three input parity functions, see Figures 6 and 7. However, as with asymmetric trees, this appears to die away as the programs become bigger.

Searching just the full trees yields a similar fitness distribution for the always-on-6 problem as for the whole search space (compare Figures 8 and 12). However the peaks corresponding to functions returning true multiples of 4, 8, 16 or 32 times are now far less prominent and instead always-on-6 itself and its compliment, always-off-6, now dominate and together represent 35% of all trees, compared to 18% when consider asymmetric trees as well. Also the troughs at odd numbers of hits are also less prominent, each representing about 0.5% rather than about 0.3% of all programs. Adding XOR to the function set (see Figure 14) has the effect of further smoothing the distribution. The peaks at either extreme are now 8% with a typical odd values near 32 being 1.4% and even being 1.8%. Both with XOR and without the distribution of the number of trues returned by full trees shows some dependence on depth of tree. However, as with even-6 parity, this appears to fade away as the programs become bigger.

4 Discussion

If we compare Figures 4, 5, 8, 9, 10 and 11 with a similar plot for the artificial ant problem on the Santa Fe trail [Langdon and Poli, 1998, Figure 2] we see in all cases a certain minimum size is required before any solutions with a certain functionality exist, and that this threshold increases with the difficulty of the functionality. Once this threshold size is exceeded the proportion of programs grows rapidly to a stable value which appears to be more-or-less independent of program size. Conversely the proportion of the search space which implements easy functionality starts high and then falls with increasing program length, again converging to a stable value which is more-or-less independent of further increases in program length. We have demonstrated this property on the 256 3-input Boolean functions, 4 6-input Boolean functions as well as the ant problem. We expect this property to hold in many cases, however demonstrating it on 261 problems is not sufficient to prove it holds in general. But it does add experimental weight to some of our claims about the nature of program fitness landscapes and their influence on the bloating phenomena [Langdon and Poli, 1997].

On average half the random trees sampled using the ramped-half-and-half method [Koza, 1992, page 93] are full. Therefore, particularly if the depth parameter is increased beyond the usual 6 (equivalent to maximum size of 63), the chances of finding at random both the even-3 and the odd-3 parity functions are considerably higher using it than using uniform search. In contrast ramped-half-and-half is less likely to find solutions to the Santa Fe ant trail problem than uniform search (see [Langdon and Poli, 1998, Table 3]). This suggests that the best method to use to create the initial random population is problem dependent.

In [Koza, 1992, Chapter 9] GP performance is shown not to be the same as random search. Indeed in the case of all but a few of the simplest problems which both GP and random search easily solve, GP performance is shown to be superior to random search. [Koza, 1992, Chapter 9] treats in detail all the 256 Boolean functions with 3 bit inputs. (See also [Lang, 1995] and [Koza, 1995]). When [Koza, 1992, page 211] compares the performance of GP with random search on these problems it explicitly assumes that programs of one length (41) are typical of the whole search space. In Section 3.1 we have verified this assumption. It should be noted that the subspaces consisting of short trees or full trees are not typical of the whole space. In particular full trees are much more likely to implement one of the parity functions than asymmetric trees which form most of the search space.

5 Conclusions

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.

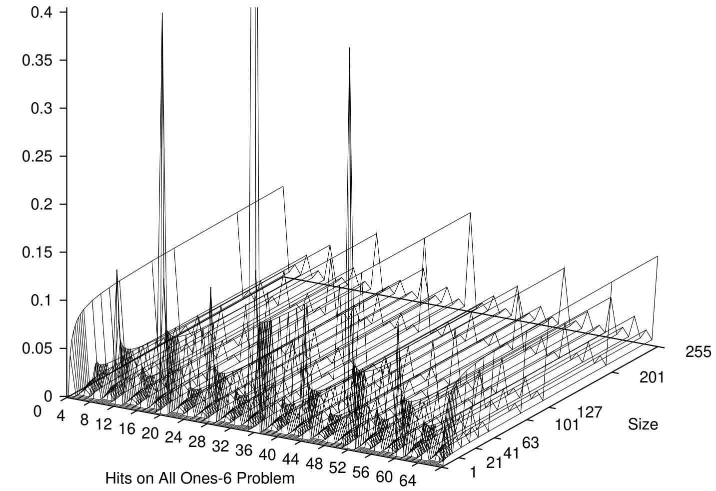

Figure 8 Number of ones returned by 6-input Boolean functions (note linear scale)

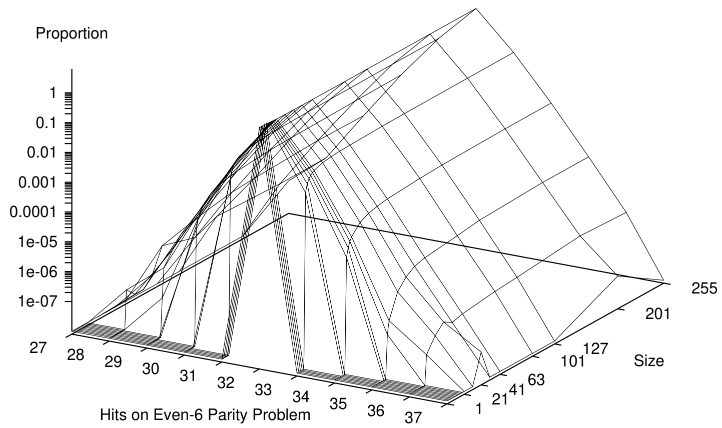

Figure 9 Even-6 parity program space

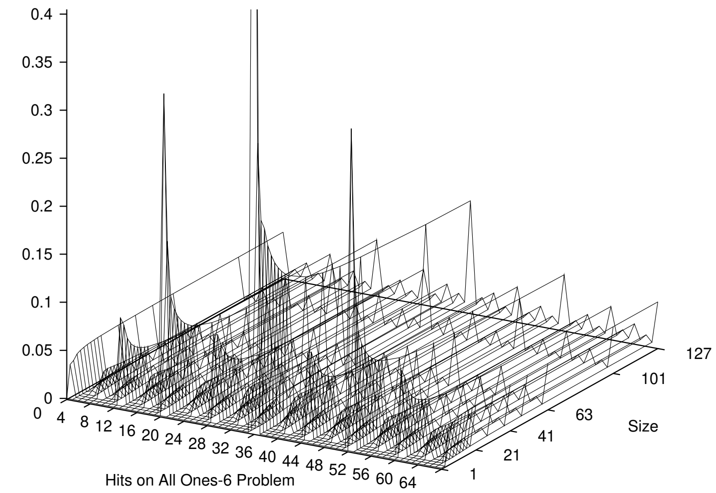

Figure 10 Number of ones returned by 6-input Boolean functions, XOR included (note linear scale)

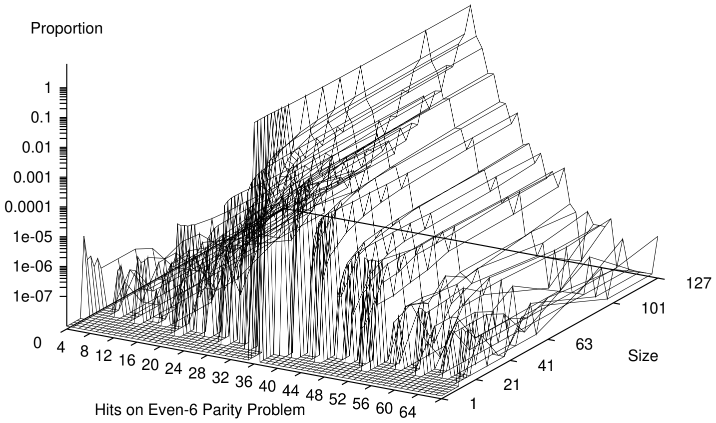

Figure 11 Even-6 parity program space, including XOR in function set

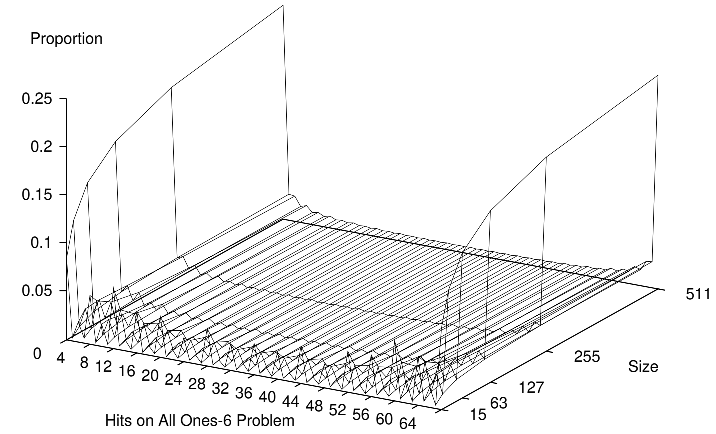

Figure 12 Number of ones returned by 6-input full trees (note linear scale)

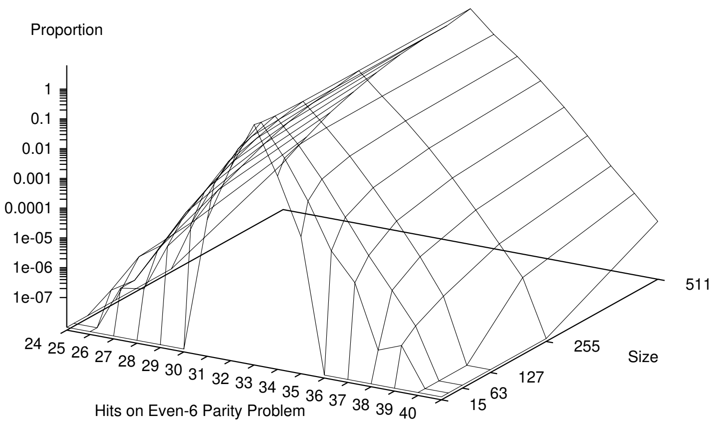

Figure 13 Even-6 parity full tree program space

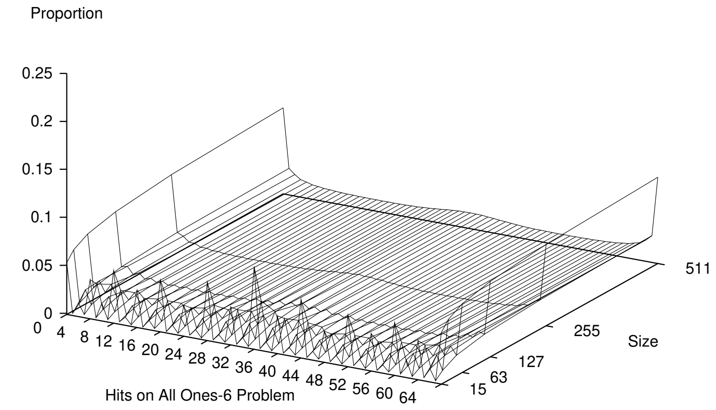

Figure 14 Number of ones returned by 6-input full trees, XOR included (note linear scale)

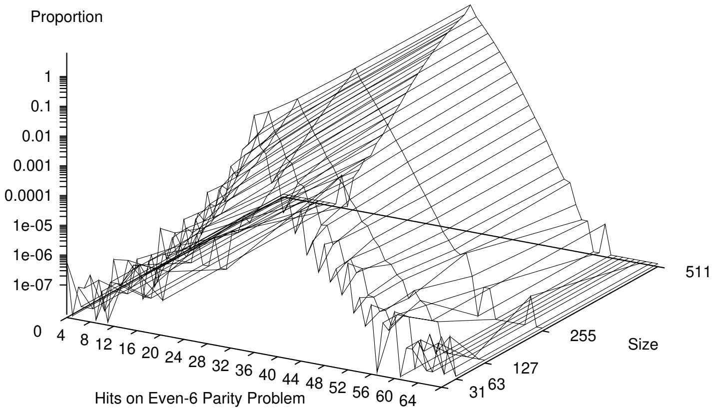

Figure 15 Even-6 parity full tree program space, including XOR in function set

Acknowledgements

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

A No Free Lunch Theorems (NFL)

Roughly speaking NFL says "the average performance of [search] algorithms across all possible problems is identical" [Wolpert and Macready, 1997, page 67]. There are only 256 (2^{2^3}) three input Boolean functions however it is not practical to use them to experimentally confirm any of the NFL theorems. This is because the number of fitness functions (i.e. problems) is too big.

In GP we consider only a small number of fitness functions, typically only one. The three input boolean functions are unusual in that 256 fitness functions have been investigated. However this is still a miniscule proportion of all the possible fitness functions and so NFL does not apply. Indeed the performance of three search algorithms (GP [Koza, 1992], hill climbing [Lang, 1995] and uniform random search) are not identical when averaged across all 256 three input Boolean functions.

Conventionally the fitness of a Boolean function of n inputs is defined as the number of times its output matches that required when tested upon all possible combinations of inputs. The total number of test patterns is 2^n. Therefore the fitness function yields a value in $0 \ldots 2^n$ and the total number of possible fitness functions is $(1 + 2^n)^{|x|}$ where $|x|$ is the size of the search space. If we consider searching in the space of trees then $|x|$ is already very big and so considering all $(1+2^n)^{|x|}$ fitness functions is infeasible. (A program to calculate $|x|$ is available from ftp node `cs.ucl.ac.uk` directory `genetic` sub-directory `gp-code` in file `ntrees.cc` and via my home page).

If instead of considering searching the space of trees, we consider search in the space of the 2^{2^n} Boolean functions themselves then there are only $(1 + 2^n)^{2^{2^n}}$ possible fitness functions. In the case of $n = 3$ this is $9^{256} = 1.93233\ 10^{244}$. Clearly comparing the performance of two or more search algorithms on this number of problems is still infeasible.

If we limit ourselves to $n = 2$, i.e. the 16 (2^{2^2}) Boolean functions of 2 bits, there are only $(1 + 2^2)^{2^{2^2}} = 5^{16} = 1.525878\ 10^{11}$ possible fitness functions. While it might be feasible to evaluate the performance of various search algorithms on them and so experimentally confirm NFL, such a demonstration is unlikely to be persuasive due to the small size of the search space. The small number of points in the search space makes it likely that stochastic search techniques such as GAs will resample one or more points. NFL explicitly requires the algorithms not to do so. Any measured difference in performance between algorithms is likely to be due to this and stochastic noise inherent in randomised search techniques.

References

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

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

[Koza, 1995] John R. Koza. A response to the ML-95 paper entitled "Hill climbing beats genetic search on a boolean circuit synthesis of Koza's". Distributed 11 July 1995 at the 1995 International Machine Learning Conference in Tahoe City, California, USA, 11 July 1995.

[Lang, 1995] Kevin J. Lang. Hill climbing beats genetic search on a boolean circuit synthesis of Koza's. In *Proceedings of the Twelfth International Conference on Machine Learning*, Tahoe City, California, USA, July 1995. Morgan Kaufmann.

[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, 1998] 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*, 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.

[Wolpert and Macready, 1997] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[1] My thanks to Mariusz Nowostawski for drawing my attention to errors on pages 1 and 3. These are corrected in this version. (Updated ftp address and tidied figures too).