Genetic Programming, Second European Workshop, EuroGP'99, Göteborg, Sweden, Riccardo Poli, Peter Nordin, William B. langdon and Terence C. Fogarty editors, 26-27 May 1999, LNCS 1598, Springer-Verlag. 22 Aug 2002[1]

Boolean Functions Fitness Spaces

W. B. Langdon[1] and R. Poli[2]

[1] Centrum voor Wiskunde en Informatica, Kruislaan 413, NL-1098 SJ Amsterdam
bill@cwi.nl http://www.cwi.nl/~bill
Tel: +31 20 592 4093, Fax: +31 20 592 4199

[2] School of Computer Science, The University of Birmingham, B15 2TT, UK
R.Poli@cs.bham.ac.uk http://www.cs.bham.ac.uk/~rmp

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 use 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 size. However the distribution of the performance of full trees is different from that of asymmetric trees and varies with tree depth.

1 Introduction

Our investigations of the artificial ant following the Santa Fe trail [6] 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 size at random its as likely to perform as well as another program of a different size 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 Sect. 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 [2, 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 (Sect. 4) and our conclusions (Sect. 5).

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 [2]. These are sufficient to construct any Boolean function but we shall also investigate including the exclusive-or function (XOR). Note [2] 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 size l [2, page 213] [1]. $|F|$ 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 size l (see Figs. 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.

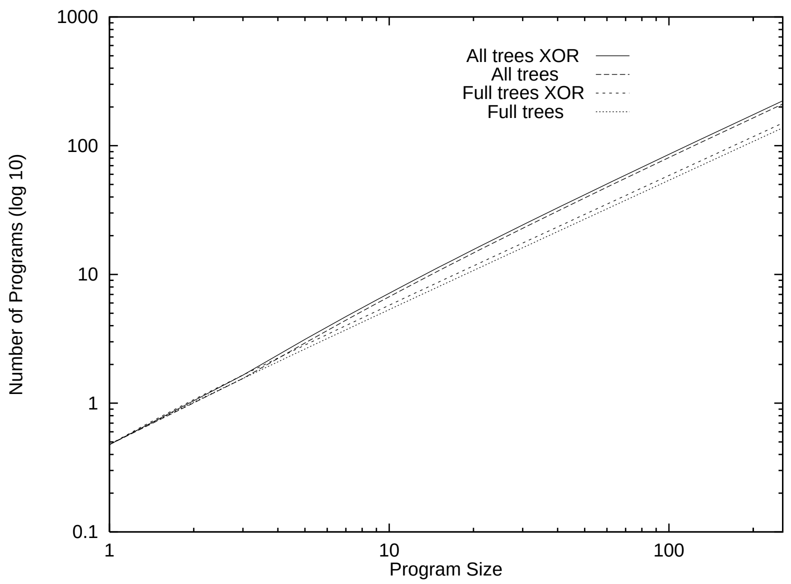

Fig. 1. Size of 3-Input Boolean Function Search Spaces (note log log scale)

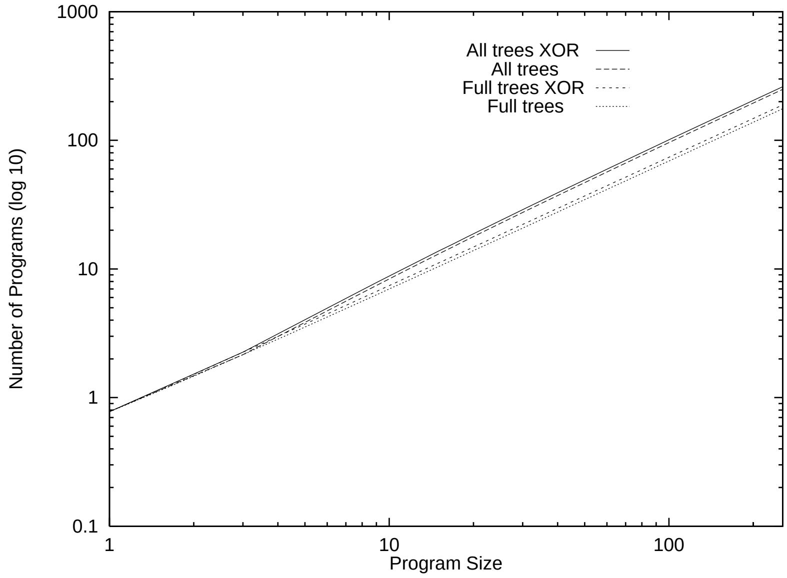

Fig. 2. Size of 6-Input Boolean Function Search Spaces (note log log scale)

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 [2, page 215].

Figure 3 shows the number of examples of each function found when 10 million trees of size 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 [2, Table 9.3]. Figure 4 shows similar plots for a wide range of sizes. (Data for trees of size 1, 3, 5, 7, 9, 11 and 13 were gathered by evaluating every tree of that size, whereas data for larger trees was gathered by randomly sampling 10 million programs for each size. C++ code to generate random programs is available via `ftp://ftp.cs.bham.ac.uk /pub/authors/W.B.Langdon/gp-code/rand_tree.cc`).

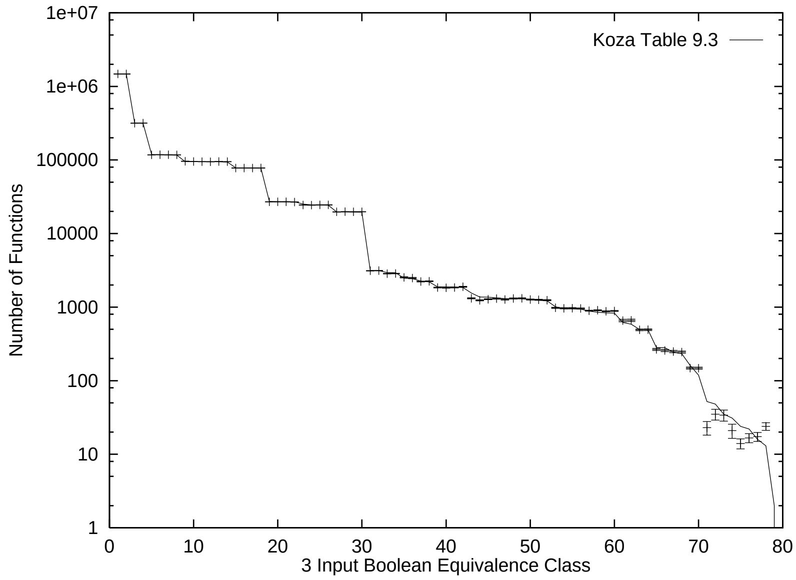

Fig. 3. Number of functions of size 41 in each equivalence class

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 equivalence 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 size 13 is no longer feasible, therefore data for size 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 Figs. 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

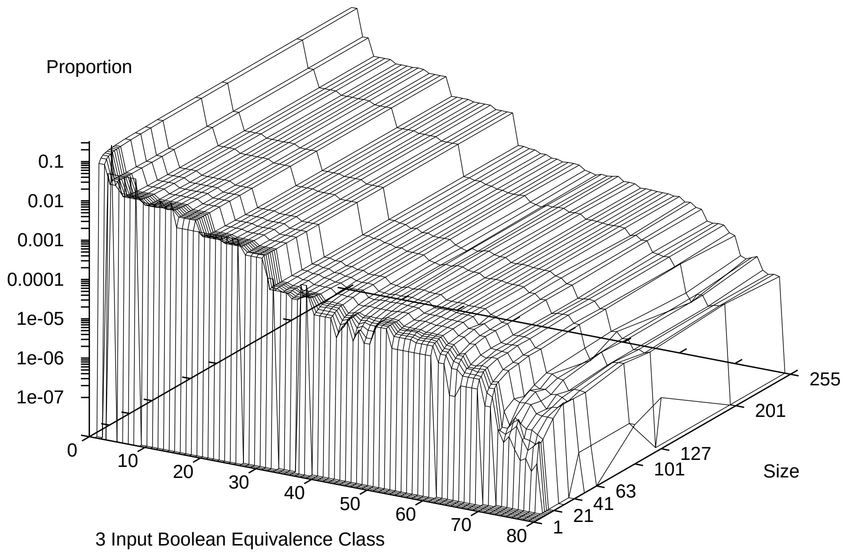

Fig. 4. Proportion of functions in each equivalence class

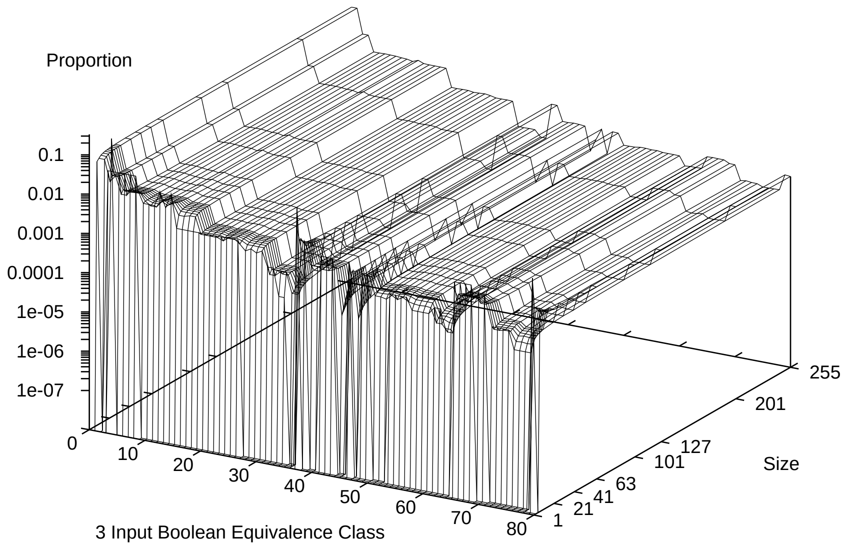

Fig. 5. Proportion of functions in each equivalence class with XOR in function set

the full trees than there are on average. When XOR is added to the function set, see Fig. 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.

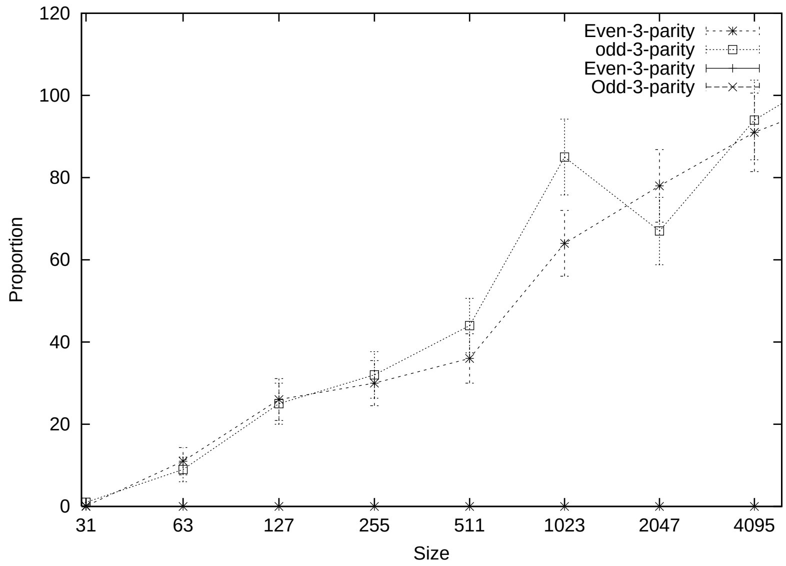

Fig. 6. 3 input parity functions in full trees

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 sizes 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 size.

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 size, only three programs (1 with 27 and two with 37) were found outside the range 28 ... 36 hits. [10, 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 size). He reports that the fitness distribution for the even-5-parity problem are even more tightly grouped in a range of 5 values. This

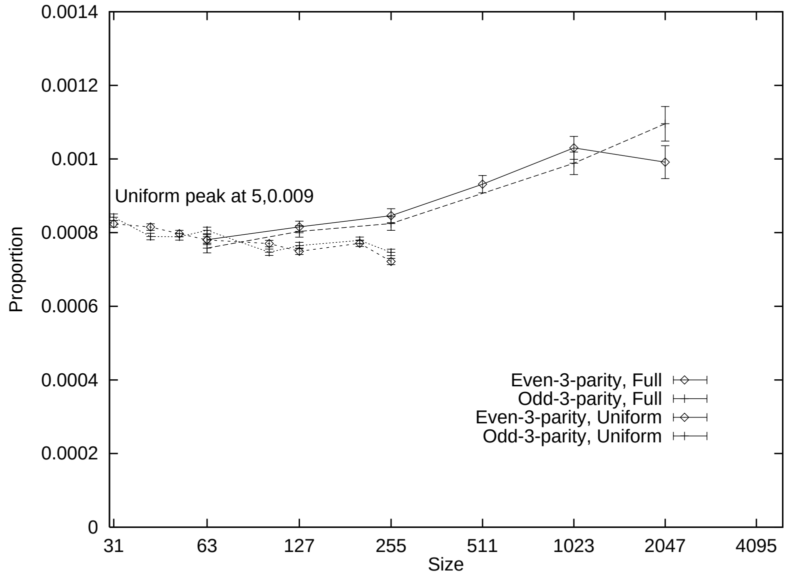

Fig. 7. Proportion of 3 input parity functions in full trees with XOR

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 Fig. 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 [10] 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 [10] 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 $\frac{n}{4}$ [10, 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 Fig. 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 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 Fig. 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.

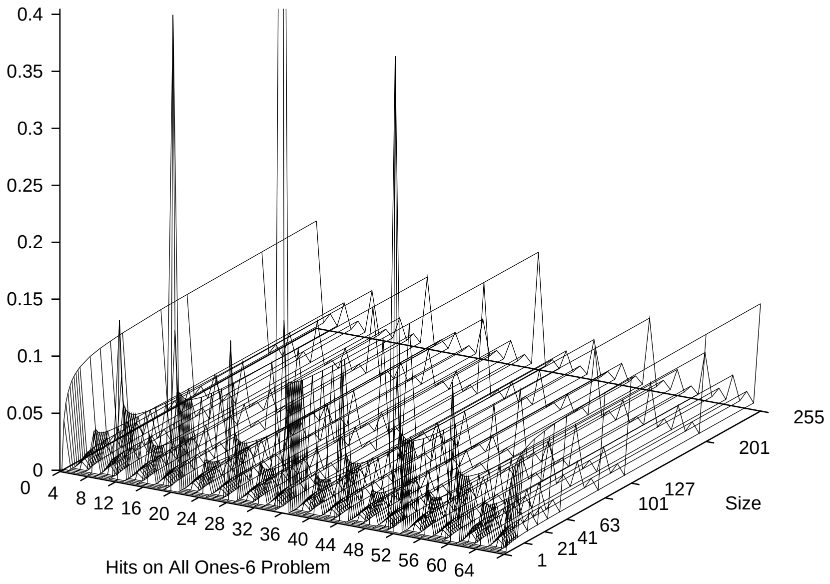

Fig. 8. Number of ones returned by 6-input Boolean functions (note linear scale)

Figure 10 shows the distribution of number of trues returned when XOR is added to the function set is a little changed (cf. Fig. 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 Fig. 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 Fig. 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 size) 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

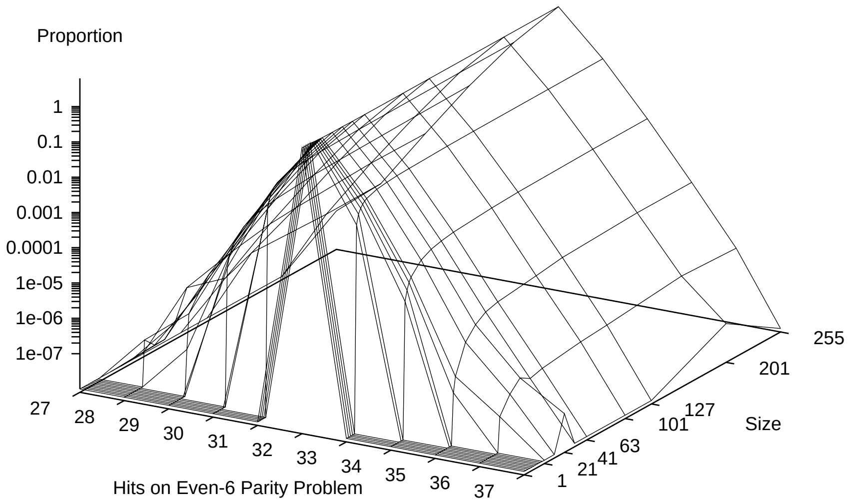

Fig. 9. Even-6 parity program space

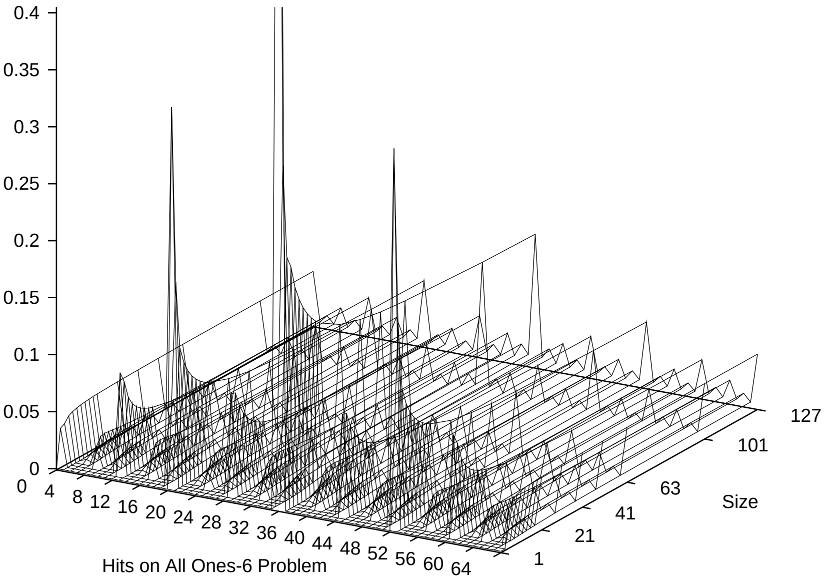

Fig. 10. Number of ones returned by 6-input Boolean functions, XOR included (note linear scale)

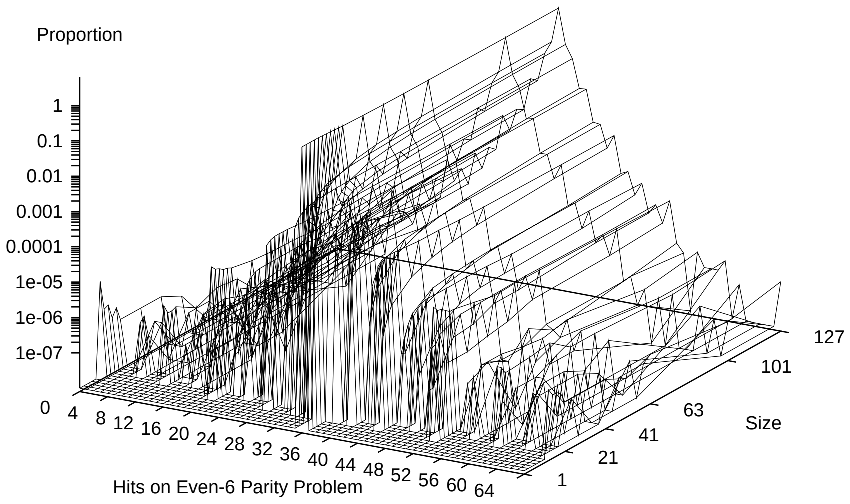

Fig. 11. Even-6 parity program space, including XOR in function set

input parity functions, see Figs. 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 Figs. 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 considering 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 Fig. 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 Fig. 4, 5, 8, 9, 10 and 11 with a similar plot for the artificial ant problem on the Santa Fe trail [6, Figure 2] and other plots for all 2, 3 and 4 Boolean functions composed on NAND gates and a continous domain problem [5], 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

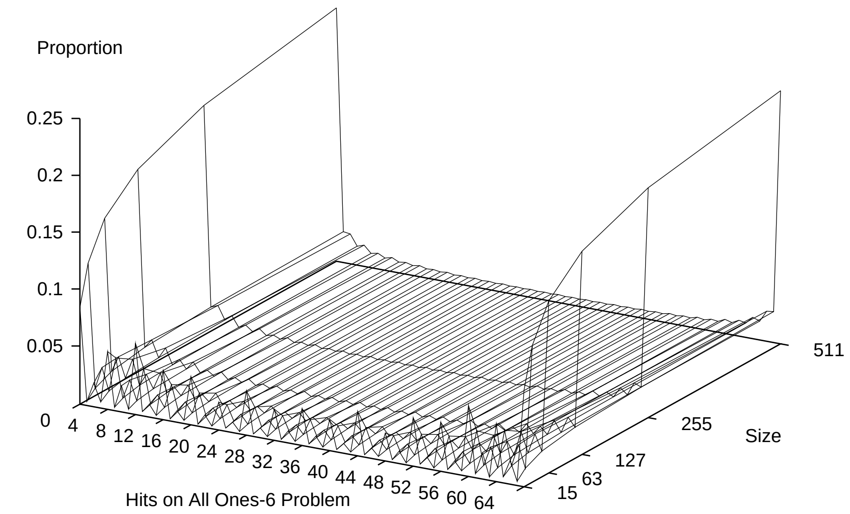

Fig. 12. Number of ones returned by 6-input full trees (note linear scale)

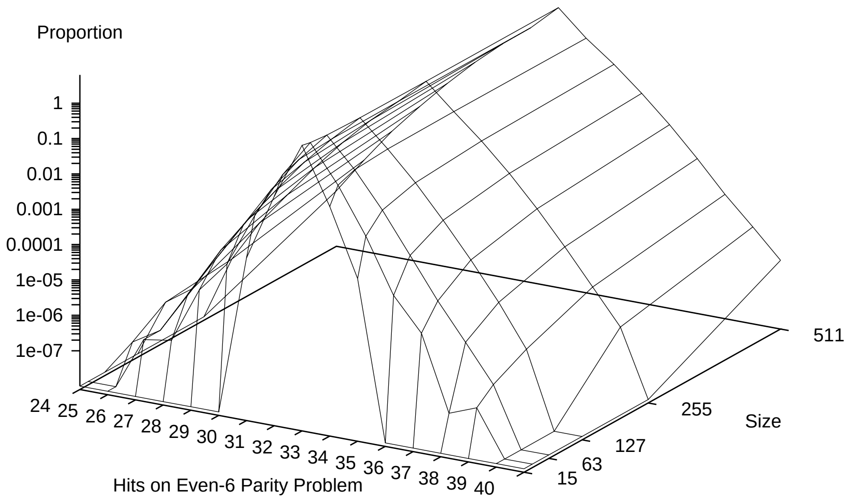

Fig. 13. Even-6 parity full tree program space

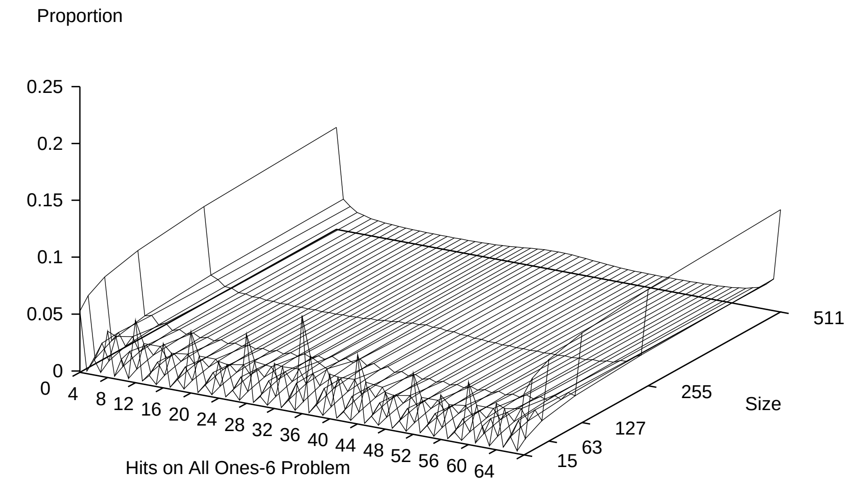

Fig. 14. Number of ones returned by 6-input full trees, XOR included (note linear scale)

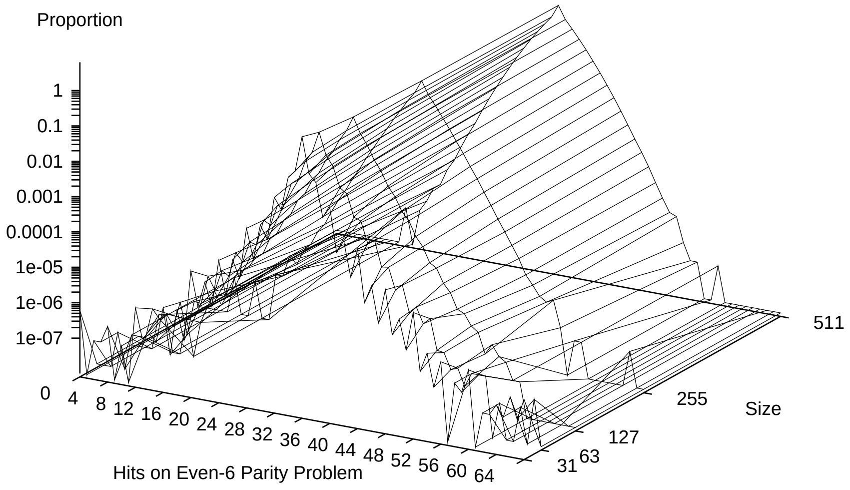

Fig. 15. Even-6 parity full tree program space, including XOR in function set

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 size, again converging to a stable value which is more-or-less independent of further increases in program size. We have demonstrated this property on the 256 3-input Boolean functions, four 6-input Boolean functions as well as the Ant problem. We expect this property to hold in many cases, indeed it can be proved in special cases [7], however demonstrating it on 261 problems (66344 including work in progress) 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 [8].

On average half the random trees sampled using the ramped-half-and-half method [2, 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 [6, Table 3]). This suggests that the best method to use to create the initial random population is problem dependent.

In [2, 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. [2, Chapter 9] treats in detail all the 256 Boolean functions with 3 bit inputs. (See also [4] and [3]). When [2, page 211] compares the performance of GP with random search on these problems it explicitly assumes that programs of one size (41) are typical of the whole search space. In Sect. 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. In the presence of code bloat where programs increase in size, the use of a depth limit (rather than size limit) may encourage the formation of full trees of the maximum permitted depth and so ease the solution of problems in which full trees contain a higher proportion of solutions while a size limit will discourage the formation of full trees and so may help in problems where the density of solutions is lower in full trees.

If we are right and the density of solutions changes little with program size then there is no intrinsic advantage in searching programs longer than the threshold. Of course, in general, we will not know in advance where the threshold is. Also it may be that some search techniques perform better with longer programs, perhaps because together they encourage the formation of smoother more correlated or easier to search fitness landscapes [9]. However in practice searching at longer sizes is liable to be more expensive both in terms of memory and also time (since commonly the CPU time to perform each fitness evaluation rises in proportion to program size).

5 Conclusions

In three very different classes of problems (the Ant, Boolean and symbolic regression problems) we have now shown that the fitness space is in a gross manner independent of program size. In general the number of programs of a given size grows approximately exponentially with that size. 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.

References

1. Laurent Alonso and Rene Schott. *Random Generation of Trees.* Kluwer Academic Publishers, 1995.
2. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.
3. 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.
4. 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.
5. W. B. Langdon. Scaling of program tree fitness spaces. 31 January 1999.
6. 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.
7. W. B. Langdon and R. Poli. Why "building blocks" don't work on parity problems. Technical Report CSRP-98-17, University of Birmingham, School of Computer Science, 13 July 1998.
8. W. B. Langdon, T. Soule, R. Poli, and J. A. Foster. The evolution of size and shape. In Lee Spector, W. B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 8. MIT Press, Cambridge, MA, USA, 1999. Forthcoming.
9. Riccardo Poli and William B. Langdon. On the search properties of different crossover operators in genetic programming. 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 293–301, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.
10. Justinian P. Rosca. *Hierarchical Learning with Procedural Abstraction Mechanisms.* PhD thesis, University of Rochester, Rochester, NY 14627, February 1997.

[1] My thanks to Mariusz Nowostawski for drawing my attention to errors on pages 2 and 4. These are corrected in this version.