Why Ants are Hard

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-98-4
January 1998

Abstract

The problem of programming an artificial ant to follow the Santa Fe trail is used as an example program search space. Analysis of shorter solutions shows they have many of the characteristics often ascribed to manually coded programs. Enumeration of a small fraction of the total search space and random sampling characterise it as rugged with many multiple plateaus split by deep valleys and many local and global optima. This suggests it is difficult for hill climbing algorithms. Analysis of the program search space in terms of fixed length schema suggests it is highly deceptive and that for the simplest solutions large building blocks must be assembled before they have above average fitness. In some cases we show solutions cannot be assembled using a fixed representation from small building blocks of above average fitness. These suggest the Ant problem is difficult for Genetic Algorithms.

Random sampling of the program search space suggests on average the density of global optima changes only slowly with program size but the density of neutral networks linking points of the same fitness grows approximately linearly with program length. This is part of the cause of bloat.

Previously reported genetic programming, simulated annealing and hill climbing performance is shown not to be much better than random search on the Ant problem.

1 Introduction

There have often been claims that automatic programming is hampered by the nature of program spaces. These are undoubtedly large [Koza, 1992, page 2] and, it often claimed, badly behaved with little performance relationship between similar programs [O'Reilly, 1995, page 8]. In this paper we present a systematic exploration of the program space of a commonly used benchmark problem (Sections 2 and 3).

In Section 4 we calculate the number of fitness evalutions required by two types of random search to reliably solve the problem and then compare this with results for genetic programming (GP) and other search techniques. This shows most of these techniques have broadly similar performance, both to each other and to the best performance of totally random search.

This prompts us to consider the fitness landscape (Section 5) and schema fitness and building blocks (Section 6) with a view to explaining why these techniques perform badly and to find improvements to them. In Section 7 we described the simpler solutions. Their various symmetries and redundancies mean essentially the same solution can be represented in an unexpectedly large number of different programs. Finally in Section 8 we consider why the problem is important and how we can exploit what we have learnt and in Section 9 we give our conclusions.

2 The Artificial Ant Problem

The artificial ant problem [Koza, 1992, pages 147–155] is a well studied problem often used as a GP benchmark. Briefly the problem is to devise a program which can successfully navigate an artificial ant along a twisting trail on a 32×32 toroidal grid. The program can use three operations, Move, Right and Left, to move the ant forward one square, turn to the right or turn to the left. Each of these operations takes one time unit. The sensing function IfFoodAhead looks into the square the ant is currently facing

Table 1: Ant Problem

Terminal set:	Left, Right, Move
Functions set:	IfFoodAhead, Prog2, Prog3
Fitness cases:	The Santa Fe trail
Fitness:	Food eaten
Wrapper:	Program repeatedly executed for 600 time steps.

and then executes one of its two arguments depending upon whether that square contains food or is empty. Two other functions, Prog2 and Prog3, are provided. These take two and three arguments respectively which are executed in sequence.

The artificial ant must follow the "Santa Fe trail", which consists of 144 squares with 21 turns. There are 89 food units distributed non-uniformly along it. Each time the ant enters a square containing food the ant eats it. The amount of food eaten is used as the fitness measure of the control program.

The fitness function, function and terminal sets etc. we use are identical to [Langdon and Poli, 1997a] cf. Table 1.

3 Size of Program and Solution Space

The number of different programs of a specific length is given by the size of the terminal set and the numbers of different functions in the function set of each arity (branching factor). To create a tree of a specific length a corresponding number of functions of each branching factor and number of leafs must be used. Where there are more than one branching factor available in the function set, there may be multiple combinations of function arity which give rise to a tree of the required size. In general there are multiple ways of arranging the branches and leafs. Each way gives rise to a distinct tree shape. Finally, where there are more than one terminal or more than one function of a given arity, there are multiple programs of the same shape. The number of different programs in the ant problem is plotted against their lengths in Figure 1 (and is tabulated in the "Total" row at the bottom of Table 2). As expected the number of programs grows approximately exponentially with the length of the programs. (The program used to calculate the number of programs is available via anonymous ftp from `ftp.cs.bham.ac.uk` in `pub/authors/W.B.Langdon/gp-code/ntrees.cc`).

For the shorter programs it is feasible to explore the program space exhaustively. Table 2 summarises the programs space up to programs of length 14. Table 2 shows the program space is highly asymmetric with almost all programs having very low scores and the proportion with higher scores falling rapidly (but not monotonically) to a low point near 72. Above 72 it rises slightly. The modal score is zero, the median is one and the mean rises with length from 1 to 2.7 while the standard deviation remains near 4 (cf. Figure 7). There is some dependence upon program length and, as expected, programs must be above a minimum size to reach modest scores. However above the minimum size the number of programs with a given score rises rapidly, being a roughly constant proportion of the total number of programs. There are an unexpectedly high number of solutions (albeit a tiny fraction of the total) and their number similarly grows with program size.

For longer programs exhaustive search is not feasible and instead we sampled the program space randomly in a series of Monte Carlo trials for a number of program sizes. For each such size between 10,000,000 random programs where generated and tested. The random programs where chosen uniformly from the set of possible programs of the specific length using the bijective random tree creation algorithm described in [Alonso and Schott, 1995, Chapter 4]. Where there are multiple different combinations of function arity which give rise to trees of the required size, one was chosen at random in proportion to the number of trees it contains. Each tree was converted to a program by labelling each of its nodes with a function or terminal of the correct arity chosen uniformly at random from those that match in the function or terminal set. In this way we ensure every program of the specified length has the same chance of being chosen.

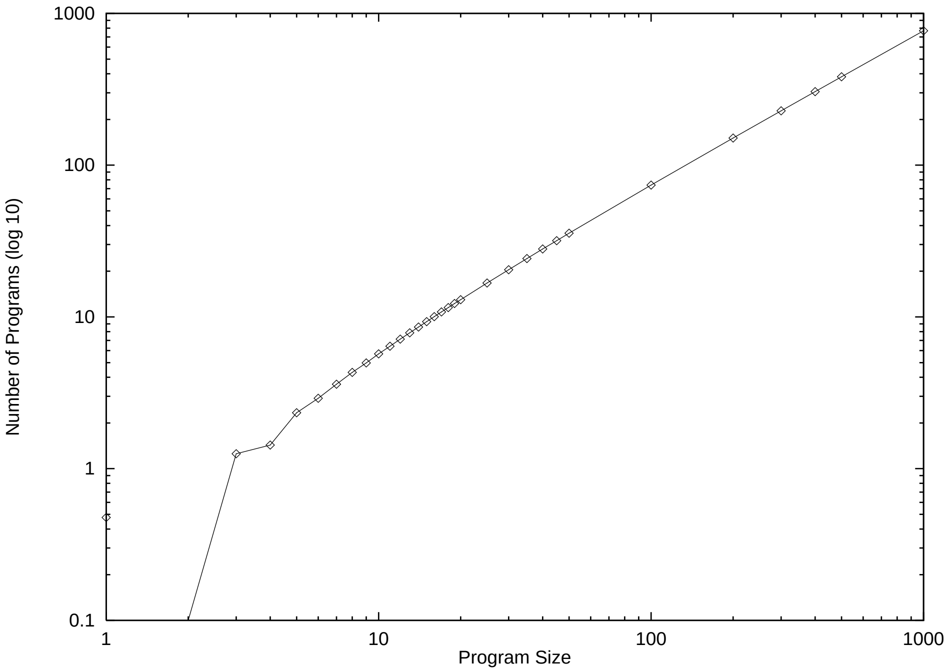

Figure 1: Number of programs of a specific length (note log log scale)

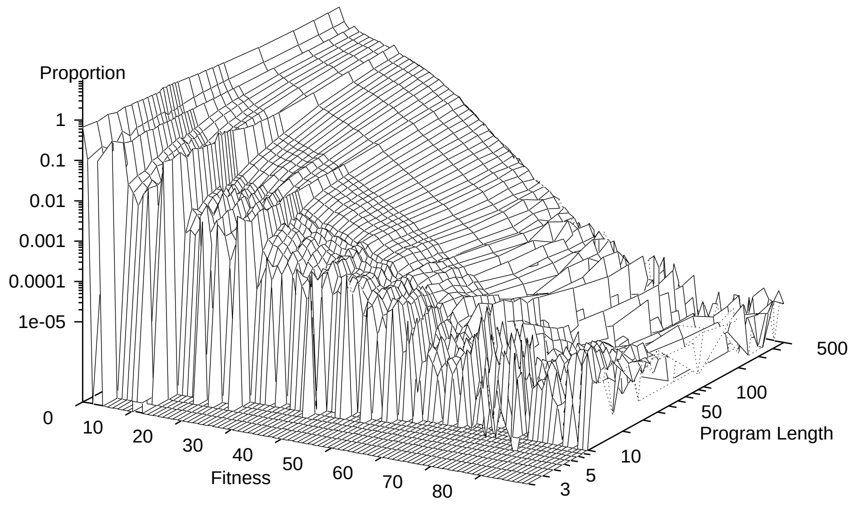

Figure 2: Proportion of programs of a given length by their fitness. Values for lengths 15 and above are based on Monte Carlo sampling.

Table 2: Number of Trees and Distribution of Fitness

	Length													
Score	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	2	0	12	16	136	423	2262	10452	49088	252803	1227679	6443754	32595908	171997308
1	0	0	2	4	18	150	449	3058	13806	77269	380979	2070276	10954364	58002558
2	0	0	0	1	3	25	112	909	3429	21902	123174	646831	3587432	19987018
3	1	0	2	6	31	96	530	2779	11736	63996	318817	1656409	8501211	45339974
4	0	0	0	0	3	14	72	527	2526	16250	86999	501521	2820383	15927690
5	0	0	0	0	2	10	58	417	1844	13047	67895	390963	2213475	12466189
6	0	0	0	0	1	8	25	177	1155	6826	33174	216479	1248818	6766377
7	0	0	0	0	2	13	35	266	1601	8076	39428	240187	1324912	6872615
8	0	0	0	0	3	10	68	412	1818	10785	56857	303276	1580134	8846059
9	0	0	0	0	0	0	2	28	183	1392	8218	57485	348331	2053040
10	0	0	0	0	1	2	18	76	461	2758	12465	75079	406998	2276683
11	0	0	2	0	16	51	297	907	5876	27403	120960	659392	3245735	16642082
12	0	0	0	0	0	0	3	52	190	1326	8296	45293	258390	1525769
13	0	0	0	0	0	0	4	40	154	1203	8011	44681	266859	1548594
14	0	0	0	0	0	0	3	24	105	770	5437	27113	169161	1041738
15	0	0	0	0	0	0	9	75	150	1313	11513	41711	226363	1528861
16	0	0	0	0	0	0	0	8	34	350	2584	15053	104018	654943
17	0	0	0	0	0	2	4	35	167	1108	4962	33175	197400	1078896
18	0	0	0	0	0	0	0	2	13	190	1764	11192	83119	570147
19	0	0	0	0	0	0	0	10	66	593	3028	18180	101133	660977
20	0	0	0	0	0	2	4	20	105	763	3985	25601	179522	938185
21	0	0	0	0	0	0	1	13	56	448	2501	16089	107227	581413
22	0	0	0	0	0	0	0	16	71	639	2825	21205	129354	692285
23	0	0	0	0	0	0	0	20	47	489	2372	15321	83764	509240
24	0	0	0	0	0	4	12	47	293	1723	6688	39676	194484	1048249
25	0	0	0	0	0	0	0	1	36	315	1586	10698	65746	359170
26	0	0	0	0	0	0	0	2	23	240	1785	11356	76602	379975
27	0	0	0	0	0	0	0	7	16	222	1262	8820	54491	306671
28	0	0	0	0	0	0	0	2	18	153	1049	7208	51024	258757
29	0	0	0	0	0	0	0	0	11	118	605	4705	30333	184386
30	0	0	0	0	0	0	0	3	23	203	922	6483	40544	215169
31	0	0	0	0	0	0	0	0	12	120	624	5167	37145	191740
32	0	0	0	0	0	0	0	0	2	48	387	2859	21765	129411
33	0	0	0	0	0	0	0	5	13	132	898	5315	31898	158423
34	0	0	0	0	0	0	0	2	12	99	519	3648	23694	125950
35	0	0	0	0	0	0	0	1	5	50	298	2490	20548	105190
36	0	0	0	0	0	0	0	5	12	106	349	2905	16828	98260
37	0	0	0	0	0	0	1	0	18	104	595	3189	20800	99885
38	0	0	0	0	0	0	0	2	21	183	637	4054	17473	106100
39	0	0	0	0	0	0	0	0	6	79	226	1782	8274	54753
40	0	0	0	0	0	0	0	0	8	79	267	1888	9477	56608
41	0	0	0	0	0	0	0	0	6	57	245	1682	8812	49873
42	0	0	0	0	0	0	0	5	11	92	263	1821	9083	51056
43	0	0	0	0	0	0	0	6	8	78	190	1688	7681	47354
44	0	0	0	0	0	0	0	0	0	11	109	808	4988	27119
45	0	0	0	0	0	0	0	0	0	11	84	756	3929	24649
46	0	0	0	0	0	0	0	0	6	53	184	1165	5234	30441
47	0	0	0	0	0	0	0	1	5	42	133	1068	6063	30738
48	0	0	0	0	0	0	0	0	1	14	62	532	2978	16510
49	0	0	0	0	0	0	0	0	0	4	30	293	1917	11133
50	0	0	0	0	0	0	0	0	4	41	142	970	3467	20849
51	0	0	0	0	0	0	0	0	5	38	122	748	2731	16387
52	0	0	0	0	0	0	0	0	0	2	10	160	774	6193
53	0	0	0	0	0	0	0	0	0	13	22	307	1287	8526
54	0	0	0	0	0	0	0	0	0	0	1	26	128	1568
55	0	0	0	0	0	0	0	0	0	3	6	105	340	3437
56	0	0	0	0	0	0	0	0	0	0	10	73	289	2082
57	0	0	0	0	0	0	0	0	0	0	6	58	617	2212
58	0	0	0	0	0	0	0	0	0	0	3	24	146	949
59	0	0	0	0	0	0	0	0	0	0	0	0	20	316
60	0	0	0	0	0	0	0	0	0	0	8	46	229	1790

Table 2: Number of Trees and Distribution of Fitness

	Length													
Score	1	2	3	4	5	6	7	8	9	10	11	12	13	14
61	0	0	0	0	0	0	0	0	0	0	4	30	223	1435
62	0	0	0	0	0	0	0	0	0	0	0	2	29	1113
63	0	0	0	0	0	0	0	0	0	0	5	85	285	4538
64	0	0	0	0	0	0	0	0	0	0	0	1	23	337
65	0	0	0	0	0	0	0	0	0	0	0	0	53	1610
66	0	0	0	0	0	0	0	0	0	0	0	0	3	66
67	0	0	0	0	0	0	0	0	0	0	0	15	31	435
68	0	0	0	0	0	0	0	0	0	0	0	0	3	90
69	0	0	0	0	0	0	0	0	0	0	0	0	17	2394
70	0	0	0	0	0	0	0	0	0	0	0	0	0	1063
71	0	0	0	0	0	0	0	0	0	0	0	0	3	2344
72	0	0	0	0	0	0	0	0	0	0	0	0	1	18
73	0	0	0	0	0	0	0	0	0	0	0	0	7	525
74	0	0	0	0	0	0	0	0	0	0	6	42	119	868
75	0	0	0	0	0	0	0	0	0	0	0	0	0	146
76	0	0	0	0	0	0	0	0	0	0	0	0	14	174
77	0	0	0	0	0	0	0	0	0	0	4	26	113	733
78	0	0	0	0	0	0	0	0	0	0	6	34	158	991
79	0	0	0	0	0	0	0	0	0	0	4	16	137	755
80	0	0	0	0	0	0	0	0	0	0	12	104	499	3530
81	0	0	0	0	0	0	0	0	0	0	3	64	157	2126
82	0	0	0	0	0	0	0	0	0	0	2	10	60	363
83	0	0	0	0	0	0	0	0	0	0	0	60	76	1367
84	0	0	0	0	0	0	0	0	0	0	21	188	747	5559
85	0	0	0	0	0	0	0	0	0	0	3	223	459	5734
86	0	0	0	0	0	0	0	0	0	0	0	110	173	3103
87	0	0	0	0	0	0	0	0	0	0	27	194	563	3420
88	0	0	0	0	0	0	0	0	0	0	57	399	1188	6951
89	0	0	0	0	0	0	0	0	0	0	12	48	470	2676
Total	3	0	18	27	216	810	3969	20412	95256	516132	2554416	13712490	71521461	382794984
Mean	1	0	1.7	0.9	1.7	1.9	2.0	2.06903	2.24659	2.42444	2.44969	2.59006	2.70357	2.76907
SD	3.7	0	4.4	3.8	4.4	4.4	4.4	4.45867	4.57353	4.79021	4.76974	4.94338	4.98586	5.04057
Max	3	0	11	3	11	24	37	47	51	55	89	89	89	89

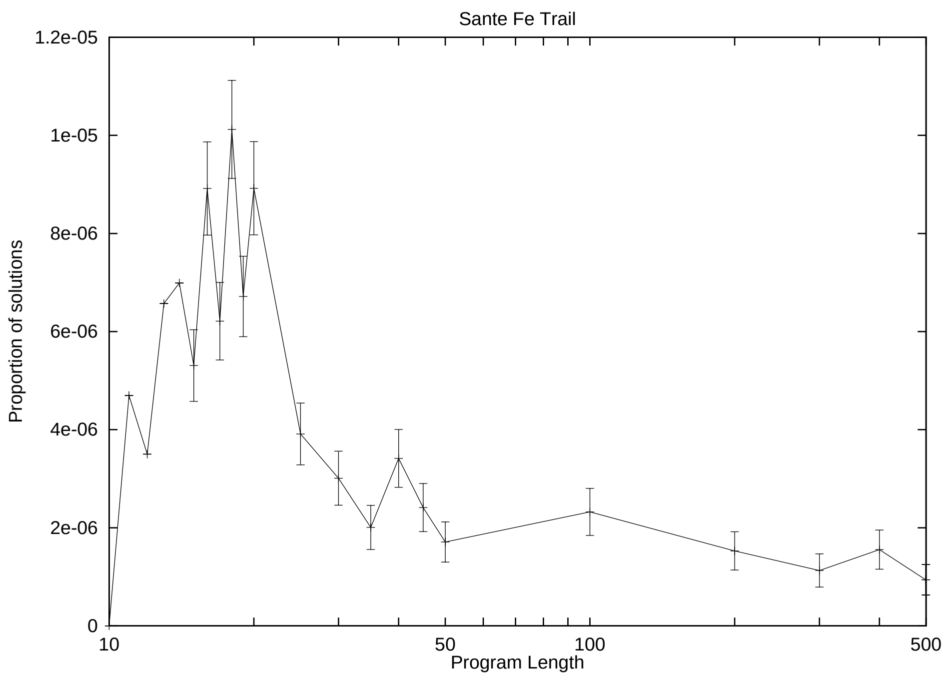

Figure 3: Proportion of programs of a given length which are solutions. Error bars indicate standard error on Monte Carlo estimates.

Figure 2 shows that the proportion of programs with a given score is approximately constant for a wide range of program lengths. Since the total number of programs rises rapidly, this means the number of programs with a given score also rises rapidly with length. This confirms assumptions in [Langdon and Poli, 1997a].

With any Monte Carlo technique there will be some stochastic error in the estimates. In the case of rare events (such as finding a solution to the ant problem) this could be large. The stochastic error was kept reasonable by using a large number of trials so a modest number of solutions were found at each length (between 9 and 101 and on average 39). An estimate of the stochastic error is plotted in Figure 3 using error bars.

4 Solution of the Ant Problem

Using the probability P of finding a solution we can calculate the number of program evaluations needed to ensure we find a solution (with probability $\geq 1 - \epsilon$). This is known as "Effort" required, cf. [Koza, 1992, page 194]:

$$\begin{aligned} E &= \frac{\log \epsilon}{\log(1-P)} \\ E &\approx -\frac{\log \epsilon}{P} \end{aligned}$$

Taking ϵ as 1% we can calculate the number of fitness evaluations E required to find at least one solution (with probability $\geq 99\%$).

4.1 Uniform Random Search

Using uniform random search and taking the maximum value for P gives us a minimum figure of 450,000 for programs of length 18. However if we allow longer programs, P falls producing a corresponding rise in E to 1,200,000 with programs of size 25 and 2,700,000 with programs of size 50 and 4,900,000 for sizes of 500. (In the ant problem as well as reducing the chance of success, longer random programs also require more machine resources to evaluate).

4.2 Ramped-Half-and-Half Random Search

Attempts to solve GP problems using random search have so far been unsuccessful [Koza, 1992]. For example in the stack problem [Langdon, 1998a, page 75] the "ramped-half-and-half" method [Koza, 1992, page 93] (which is often used to create the initial populations for GP experiments) was used to generate and test more than 49,000,000 programs random and no solutions were found.

Using the ramped-half-and-half method with a depth limit of 6, we created 20,000,000 random programs of between 3 and 242 nodes in length. Six solutions to the Ant problem were found. This gives us an estimated E figure of 15,000,000. This is higher than the corresponding figures for uniform random search, indicating in the Ant problem the bias in ramped-half-and-half leads it to search less favoured regions of the program space. For example 51% of the programs it generated contained ten or fewer nodes and thus could not be solutions to the Ant problem. (Note a high E figure need not indicate the algorithm is poor at generating initial GP populations, which are not expected to contain solutions but instead should contain a good mix of partial solutions). Another disadvantage of the ramped-half-and-half method is it will sample some program repeatedly. E.g. only five of the six solutions found are distinct from each other.

4.3 Comparison with Other Methods

Table 3 gives E values for various methods of solving the Ant problem. Rows 2–6 are from calculations in the previous sections, row 7 is from [Koza, 1992], while rows 8 onwards have been calculated from

Table 3: Effort to Solve Santa Fe Trail

Method		$E/1000$
Random (len=18)		450
Random (len=25)		1,200
Random (len=50)		2,700
Random (len=500)		4,900
Ramped-half-and-half		15,000
Koza GP	[Koza, 1992, page 202]	450
GP	[Langdon and Poli, 1997a]	450
Subtree Mutation	[Langdon and Poli, 1997b]	426
Simulated Annealing	50%–150%	748
	Subtree-sized	435
Hill Climbing	50%–150%	955
	Subtree-sized	1,671
Strict Hill Climbing	50%–150%	186
	Subtree-sized	738
Population (data for best)	50%–150%	266
	Subtree-sized [Langdon, 1998b]	390
PDGP		336

runs previously reported which used three types of varible length subtree mutation (except the last row using PDGP has not previously been reported).

From Table 3 it is clear that there are many techniques capable of finding solutions to the Ant problem and although these have different performance the best typically only do marginally better than the best performance that could be obtained with random search.

In the following sections we investigate the Ant problem fitness landscape to explain the comparitively poor performance of these search techniques.

5 Fitness Landscape

We consider two programs in the program space to be neighbours if they have the same shape and one can be obtained from the other just by changing one node. I.e. they are neighbours if making a point mutation to one program produces the other. This is the simplest neighbour relationship which means we can avoid the complications inherent in crossover operator such as GP crossover.

In the case of small programs (i.e. size 11, 12 and 13) we investigated the neighbourhoods of all the fitter programs, i.e. those with scores above 24 (in [Langdon, 1998b] in almost all runs the best individual found had a score better than 24). As expected this showed many neighbours are worse or much worse (i.e. score less than 24). It also showed that many individuals with fitness between 24 and 88 are local optima, in that none of their neighbours are fitter than them. With short programs only a few neigbours have identical fitness.

The neighbourhoods of solutions are composed of low fitness programs. For programs of length 11 or 12, apart from programs which score 24–27 or 36, all neighbours of the solutions score < 24. I.e. if a hill climber searching programs of length 11 or 12 finds a program scoring more than 36 we know it will never find a solution, without restarting. (Figure 6 shows 50 runs of a variable length representation hill climber [Langdon, 1998b] most of which became trapped at suboptimal peaks. Similar behaviour is also seen with other search techniques such as GP). There are many more solutions of length 13 and they are structurally and operationally more diverse. So their neighbourhoods are also much bigger and more diverse and include programs with scores of 24–46, 52, 54, 63, 85, 87 and 88. However five times as many have scores below 24.

For longer programs exhaustive enumeration of the landscape is not feasible and we used Monte Carlo

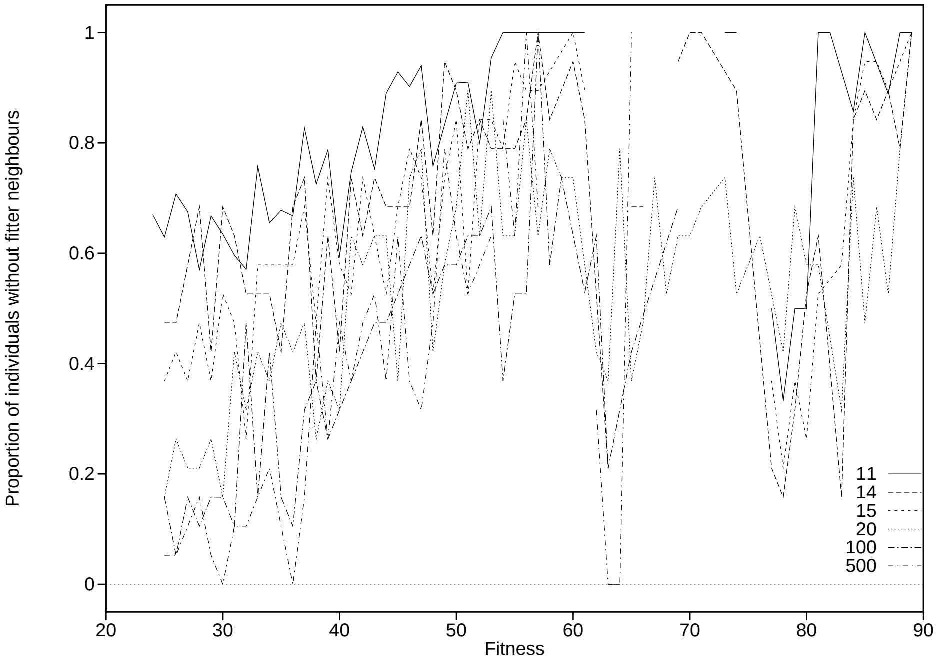

Figure 4: Proportion of programs without fitter neighbours for various program lengths. (High noise Monte Carlo estimates ignored).

sampling. As before programs of a chosen length where sampled uniformly at random. Due to the rarity of high scoring programs only a small number (up to 19) with scores 24–89 where chosen and all their neighbours where created and tested.

Figure 4 shows for most program sizes and most fitness values there are a large number of programs which do not have any fitter neighbours. In contrast Figure 5 shows the average number of neighbours with the same fitness grows with program size. These same fitness neighbours displace those that are worse, and for the longest sizes almost all programs of intermediate fitness have a large number of neighbours with the same score. (In the Ant problem a program of length l has approximately $3l/2$ neighbours).

6 Fixed Length Schema Analysis

In this section we consider the fitness of fixed length schema [Poli and Langdon, 1997] within the program space. Unlike conventional schema analysis we define a schema's fitness as the mean score for *all* programs matching the schema. From this analysis it is evident that typically there is a large variation of program scores within a schema, typically the standard deviation of score is about the same as or larger than the schema's fitness. Thus a finite sample (such as a GA population) can only reliably be used to estimate the fitness of a schema if it contains multiple independent samples from the schema. If the GA is to reliably choose between schema based on its estimate of their fitness, the number of samples (i.e. programs) must be even bigger where the schema have similar fitness. (Of course the assumption that programs are independent is not justified after selection etc.)

The competetion between schema can be viewed as a heirarchy of competetions. The outermost level being between different lengths, then between different shapes of the same length (hyperspaces) and finally between different schema of the same shape. Figure 7 shows the Ant problem is difficult at the outermost level. The region containing the highest concentration of solutions (length=18) has a fitness of 2.9 but longer programs are on average fitter than this. While the standard deviation is large compared to the mean, a typical initial GP population is likely to be large enough to be able to reliably prefer

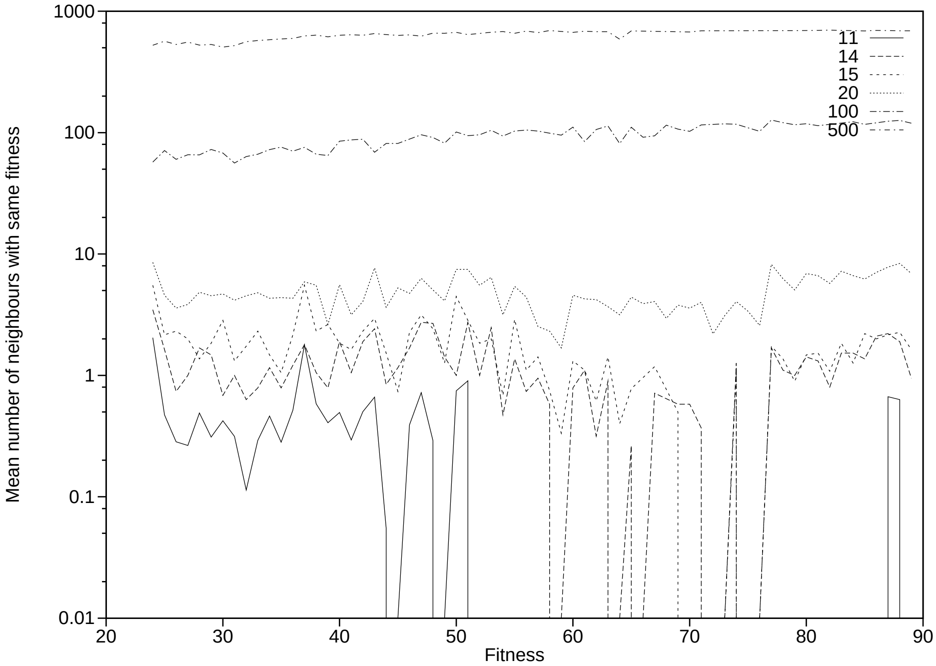

Figure 5: Mean number of neighbours with same score for various program lengths.

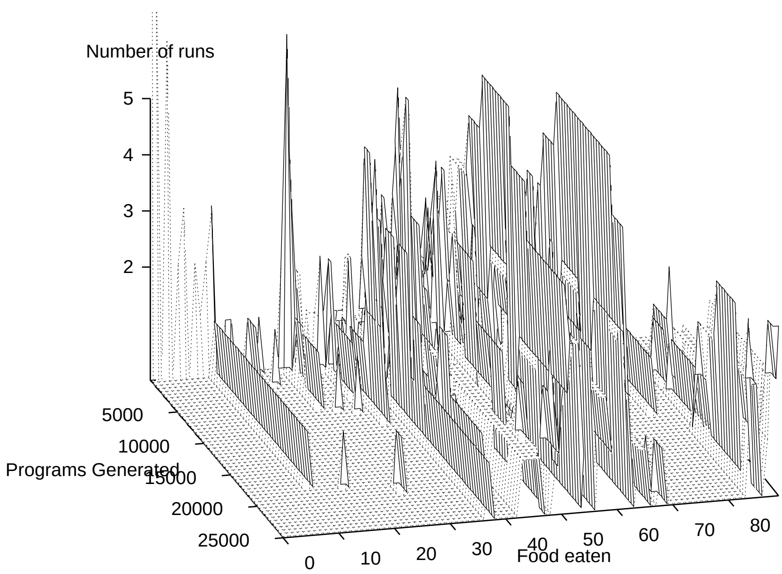

Figure 6: Evolution of best fitness in 50 hill climbing runs using 50%–150% tree mutation

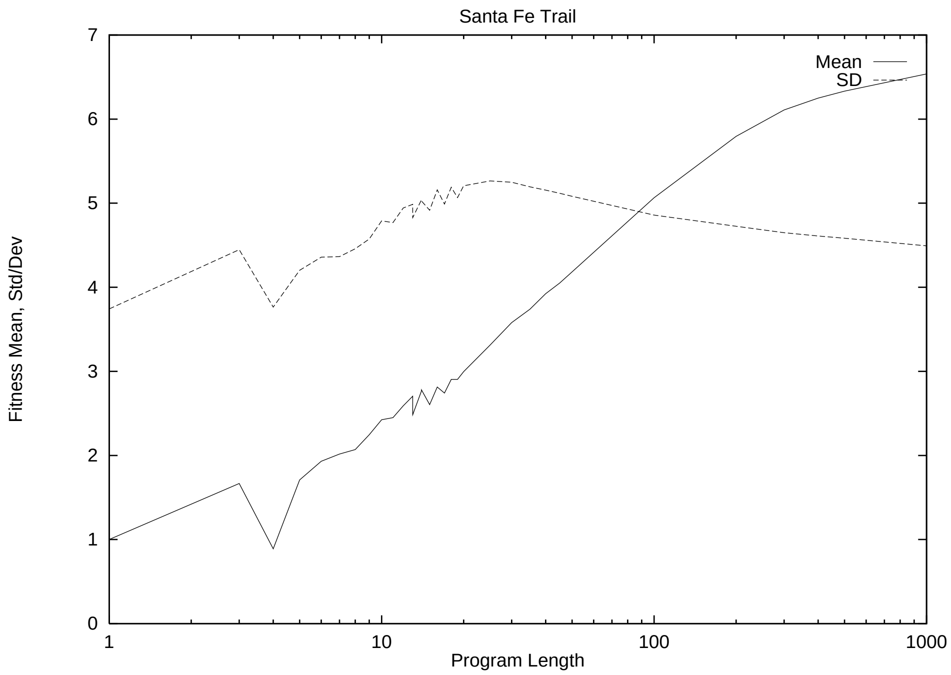

Figure 7: Mean and Standard Deviation of program scores vs. length

solutions longer than 18 over those of length 18.

Figure 8 shows the distribution of schema fitness for one of the hyperspaces containing solutions of length 11. (The other two hyperspaces are similar). Looking at the order zero schema, i.e. the hyperspace, we see it has a fitness above the average for programs of the same length, however there are other hyperspaces which are fitter.

Comparing schema of the same order we see, apart from order 9 (i.e. programs) there are always schema outside the hyperspace with higher fitness. However within the hyperspace, for a given order, the fittest schema is always one containing a solution. (Prog3 is the only function which takes three arguments. If a program's shape is given and it contains three way branches, then there must be a Prog3 at these points. I.e. the location of Prog3 are fixed. Therefore we exclude Prog3 from the schema order). It was feasible to consider all schema of order 2 or less. As Figure 8 shows there are many schema which do not contain a solution which are fitter than many of the same order that do. Also there are small components of solutions with below average fitness. It is not until more than five components have been assembled that all schema containing solutions have above average fitness. I.e. using a fixed representation solutions of length 11 cannot be assembled from small building blocks (low order schema) of above average fitness.

Turning to programs of length 12, cf. Figure 9, looking at order zero we see a similar picture to length 11: hyperspaces containing solutions have fitness above the average for programs of the same length, however there are other hyperspaces which are fitter.

Comparing schema of the same order we see, apart from order 11 (i.e. programs) there are always schema outside the hyperspace with higher fitness. Also (unlike length 11) within the hyperspace the fittest schema of each order for orders 4–8 does not contain a solution. As with length 11, there are many schema which do not contain a solution which are fitter than many of the same order that do. Also there are small components of solutions with below average fitness. It is not until more than six components have been assembled that all schema containing solutions have above average fitness.

Turning to programs of length 13 the situation is complicated by the much larger number of solutions and their diverse nature. We have selected three hyperspaces which contain solutions of different types, cf. Figures 10, 11 and 12. (Only data for schema of order, 0, 1 and 2 is available).

Looking at order zero we see a similar picture to length 12: the three selected hyperspaces have fitness

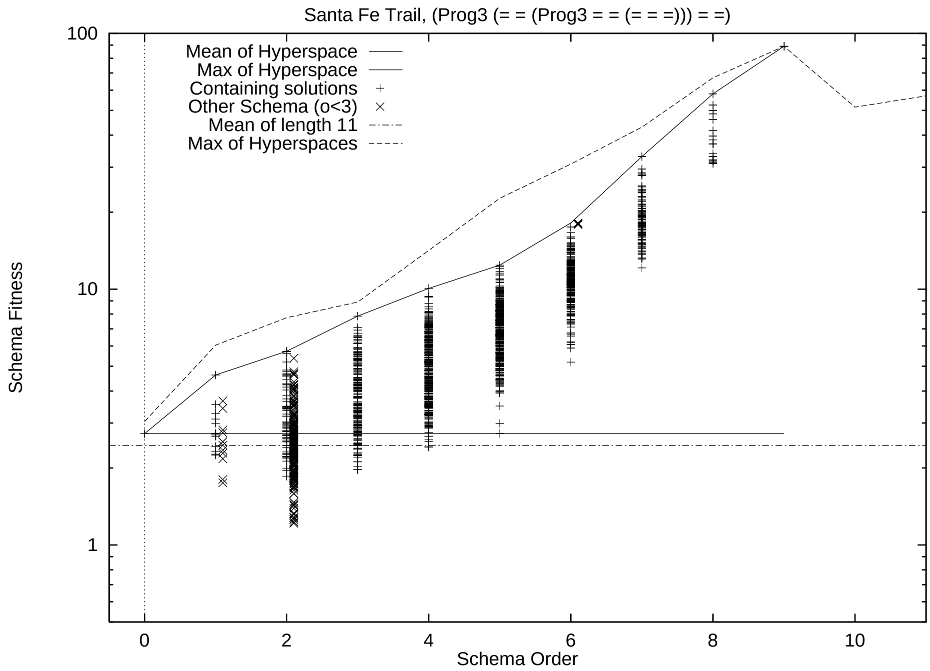

Figure 8: Distribution of Schema fitness within a Hyperspace of length 11 containing a solution

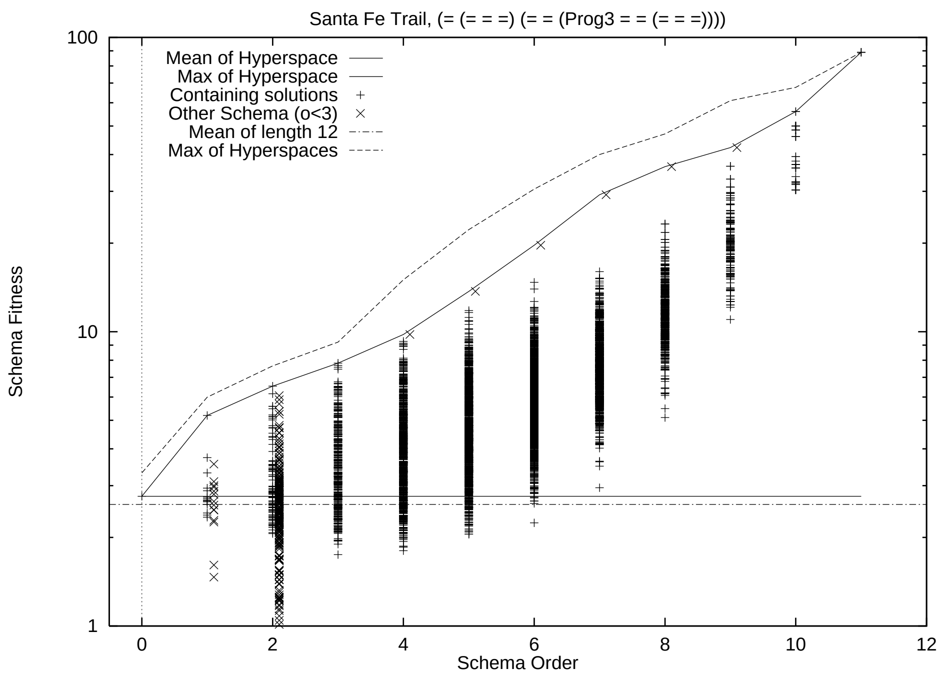

Figure 9: Distribution of Schema fitness within a Hyperspace of length 12 containing a solution

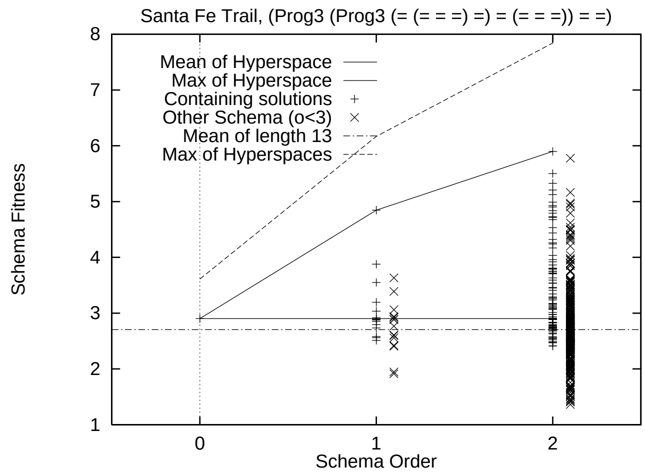

Figure 10: Distribution of Schema fitness within a Hyperspace of length 13 containing a (two move) solution

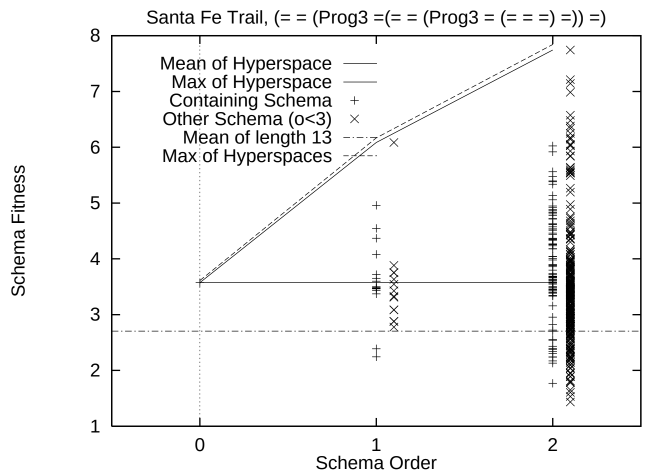

Figure 11: Distribution of Schema fitness within a highest fitness Hyperspace of length 13 containing a solution

above the average for programs of the same length, however (apart from the hyperspace chosen because it has the highest fitness) there are other hyperspaces which are fitter.

Comparing schema of the same order we see there are always schema outside the hyperspace with higher fitness (although the difference is small for the fittest hyperspace). As with lengths of 11 and 12, in two hyperspaces the fittest schema of order 0, 1, and 2 contain a solution (cf. Figures 10 and 12). However in the fittest hyperspace (Figure 11) the fittest schema of order 1 and 2 do not contain solutions. As with lengths 11 and 12, there are many schema which do not contain a solution which are fitter than many of the same order that do. Also there are small components of solutions with below average fitness.

7 The Solutions

We have analysed the shorter solutions (cf. Figures 13 to 16). As we shall see all the solutions of length 11 and 12 and most of those of length 13 are variations of each other.

Figure 13 shows all the solutions of length 11. There are twelve solutions of length 11. Not only

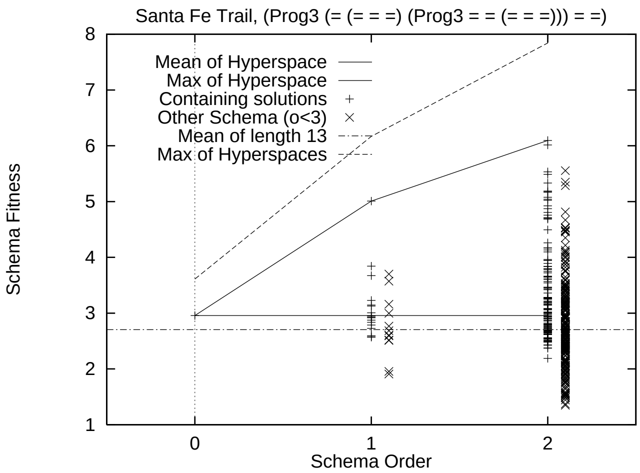

Figure 12: Distribution of Schema fitness within a Hyperspace of length 13 containing a (intron) solution

are they genetically distinct but they cause different behaviour by the ant. However we can recogise certain symmetries. For example they contain pairs of ant rotate operations and it is no surprise that these can be either pairs of Left or pairs of Right terminals. Another symmetry is that the program consists of three parts which have to be performed in order but the ant can start with any one of the three and still traverse the trail. Since the solution codes each of these as an argument of the root, the root's arguments can be rotated. Each rotation gives rise to a genetically different program, with slightly different behaviour. Each gives rise to a different tree shape and so the 12 solutions lie in three distinct hyperspaces.

The solutions of length 12 are the same as those of length 11. They are made one node longer by replacing a single Prog3 function with two Prog2 (cf. Figure 14). There are a total of four ways of doing this for each solution of length 11 giving rise to 48 solutions of length. While these are genetically distinct from each other and the solutions of length 11 they represent identical behaviour. There are 12 tree shapes (hyperspaces) each containing 4 solutions.

Extending this we can see that there must also be 48 solutions of length 13 created by replacing both Prog3 with Prog2 (there are four ways of arranging the Prog2). However there are other ways to make use of the available space to represent the same solutions. This is done by adding introns. Each of the non-Prog3 nodes can be replaced by an IfFoodAhead one of whose arguments is the previous node (and its arguments) and the other is either a terminal which is identical to the other argument or is never executed (cf. Figure 15). Most solutions of length 13 are of this type.

Thirteen nodes allow solutions of a different type which consecutively performs two moves before looking for food (cf. Figure 16). Again there is symmetry in that the ant can be rotated either to the right or to the left but whichever is done first the opposite must be done in the later part of the program. This give rise to programs of the same shape with the same score. The program now consists of five parts which have to be executed in the correct order but, as with solutions of length 11, it does not matter which is first. Each of these five orderings gives rise to a different behaviour but each traverse the trail. (However they take slightly different amounts of energy to do so. Including energy as part of the fitness measure would give a means of breaking the symmetry of these solutions). Additionally there are three ways to arrange the arguments of the two Prog3 which are functionally identical. Each of these rearrangements yields solutions of different shapes.

Most of the other solutions of length 13 also perform two consecutive Move operations. These and the remaining solution of length 13 have less symmetry and are fewer in number.

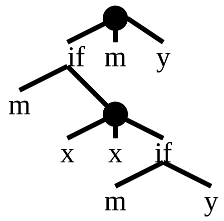

Figure 13: Solutions of length 11. x and y can be either Left or Right and the three arguments of the root can be rotated, giving 12 solutions.

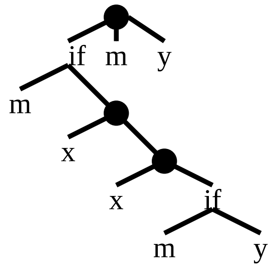

Figure 14: Solutions of length 12. Like solutions of length 11 (x and y can be either Left or Right, the three arguments of the root can be rotated) additionally one Prog3 is replaced by two Prog2 giving 48 solutions.

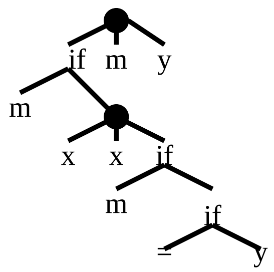

Figure 15: Intron solution of length 13. Like solutions of length 11 (x and y can be either Left or Right, the three arguments of the root can be rotated) and the = can be any terminal as it is never executed.

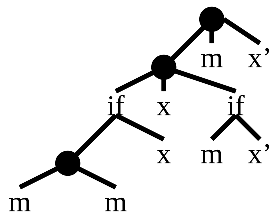

Figure 16: Solutions of length 13 performing two Moves. x can be either Left or Right but then x' must rotate in the opposite direction, again the arguments of the root can be rotated and there are equivelent ways to order the arguments of the two Prog3.

8 Discussion

From the previous sections it is clear that the Ant problem has the features often suggested of real program spaces. The program space is large and, using the simplest neighbour relationship, forms a Karst landscape containing many false peaks and many plateaus riven with deep valleys. It is clear from an analysis of the simplest solutions that there are multiple distinct and conflicting solutions to the problem, some arising from symmetries in the primitive set and some from the problem itself. The landspace is riddled with neutral networks linking programs of the same fitness in a dense and suffocating labyrinth.

A limited analysis of the schema indicates the problem is deceptive at all levels. Longer programs are on average slightly fitter but contain a slightly lower density of solutions. There are hyperspaces which do not contain solutions which are fitter than those of the same length which do. There are low and middle order schema which are required to build solutions but which are below average fitness. Schema typically have a high fitness variance. This means practical sized samples give noisy estimates of their fitness, leading GAs to choose between them randomly. However the fitness of low order schema may be estimated more reliably (as GA populations can contain many instances of them). Where they are deceptive, this may lead a GA to discard them. (Extinction of complete primitives was seen in the list and stack problems [Langdon, 1998a, Chapter 6 and 8]). If real program spaces have these characteristics (we expect them to do so but be still worse) then it is important to be able to demonstrate scaleable techniques on such problem spaces. The Santa Fe trail provides a tractable problem for such demonstrations. From Table 3 it is obvious that current techniques are not doing well on it.

We have only considered the simplest solutions using a fixed representation but we have shown they cannot be assembled from small components of above average fitness (i.e. building blocks). Indeed many constructs which a human programmer might use when constructing solutions have below average fitness. However it is possible building blocks of above average fitness which GP uses exist and longer solutions can be constructed from them. Their assembly may be eased by exploiting the variable length of the representation.

Current GP techniques are not exploiting the symmetries of the problem. These lead to essentially the same solutions appearing to be the opposite of each other. Eg. either a pair of Right or pair of Left terminals at a particular location may be important. If the search technique does not recognise them as the same thing it may spend a lot of effort trying to decide between them, when perhaps either would do (cf. "competing conventions" in artificial neural networks). A possibly useful approach is to break this symmetry (e.g. by putting more of one primitive in the initial population) to bias the technique so that it chooses one option quickly. The tangled network of programs with same fitness which consumes much machine resources by promoting bloat might also be addressed by introducing a small bias. In the Ant problem we would expect a slight bias in favour of shorter programs to be beneficial as solutions are more frequent when programs are short.

It is clear the Ant problem is essentially difficult because of the large number of local optima. These are created by the combination of the representation, the neighbour operator and the fitness function. While there may be improvements to the representation or better search techniques we should also consider the fitness function, particularly how we reward partial solutions.

9 Conclusions

We have started an examination of the program space of a GP benchmark problem. We have shown that there are many distinct solutions to the problem and the density of solutions in the program space is unexpectedly high. Indeed genetic programming and other search techniques do not perform enormously better than random search. Using the program landscape and schema analysis we have shown why the artificial ant following the Santa Fe trail problem is difficult for these search techniques and these suggest reasons why the Ant problem may be indicative of real problem spaces and so be worthy of further study.

Acknowledgements

This research was funded by the Defence Research Agency in Malvern and the authors would like to thank members of the EEBIC group.

References

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

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

[Langdon and Poli, 1997a] 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, 1997b] W. B. Langdon and R. Poli. Fitness causes bloat: Mutation. In C. Clack, K. Vekaria, and N. Zin, editors, *ET'97 Theory and Application of Evolutionary Computation*, pages 59–77, University College London, UK, 15 December 1997.

[Langdon, 1998a] W. B. Langdon. *Data Structures and Genetic Programming.* Kulwer, 1998. Forthcoming.

[Langdon, 1998b] W. B. Langdon. Program growth in simulated annealing, hill climbing and populations. In *1998 IEEE International Conference on Evolutionary Computation*, 5-9 May 1998. Forthcomming.

[O'Reilly, 1995] Una-May O'Reilly. *An Analysis of Genetic Programming.* PhD thesis, Carleton University, Ottawa-Carleton Institute for Computer Science, Ottawa, Ontario, Canada, 22 September 1995.

[Poli and Langdon, 1997] Riccardo Poli and W. B. Langdon. A new schema theory for genetic programming with one-point crossover and point mutation. In John R. Koza *et al.* editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 278–285, 13-16 July 1997. Morgan Kaufmann.

[Poli, 1997] Riccardo Poli. Evolution of graph-like programs with parallel distributed genetic programming. In Eric Goodman, editor, *Genetic Algorithms: Proceedings of the Seventh International Conference*, 19-23 July 1997. Morgan Kaufmann.