Solving the Artificial Ant on the Santa Fe Trail Problem in 20,696 Fitness Evaluations

Steffen Christensen, Franz Oppacher
School of Computer Science, Carleton University
1125 Colonel By Drive, Ottawa, Ontario, Canada, K1S 5B6
Tel: 1-613-520-4333

idyll@rogers.com, oppacher@scs.carleton.ca

ABSTRACT

In this paper, we provide an algorithm that systematically considers all small trees in the search space of genetic programming. These small trees are used to generate useful subroutines for genetic programming. This algorithm is tested on the Artificial Ant on the Santa Fe Trail problem, a venerable problem for genetic programming systems. When four levels of iteration are used, the algorithm presented here generates better results than any known published result by a factor of 7.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning --- *Induction*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search --- *Heuristic methods*.

General Terms

Algorithms, Performance, Experimentation

Keywords

Genetic Programming, Representations, Running Time Analysis, Speedup Technique

1. INTRODUCTION

The idea of automatically generating subroutines for genetic programming has been around for a long time, at least since Koza's automatically defined functions [1]. In this paper, we provide an algorithm for systematically attempting all small trees for a given problem to automatically generate useful subroutines. We test this algorithm on the Artificial Ant on the Santa Fe Trail problem [2]. This problem is of interest not only because it is an extensively used test problem, but because it has been subjected to intensive analysis by Langdon and Poli in [3]. In their Figure 3, the authors demonstrated that there are proportionately more solutions of small size than of large size. In this paper, we use this idea, combined with the knowledge that there are exponentially fewer small trees than large trees, to systematically consider all small trees as candidates for subroutine generalization. We here describe a progressive algorithm that moves systematically through the best trees of a given size and considers them as candidates for subroutine generation. We demonstrate this algorithm on the Santa Fe Trail problem, using the summary table in [3] as a reference for the best published performance. Compared to the results of Chellapilla [4] who achieved a computational effort of 136,000 fitness evaluations, our algorithm improves upon it sevenfold.

GECCO'07, July 7–11, 2007, London, England, United Kingdom.
Copyright 2007 ACM 978-1-59593-697-4/07/0007...$5.00.

2. IMPROVING ON STANDARD GENETIC PROGRAMMING

2.1 Fairly Evaluating the Work of Enumeration

As Langdon and Poli describe in [3], and Luke and Panait consider in [5], we can uniformly generate trees of a given size using the simple random tree-generation algorithm of Iba [6]. Let *trees*(*n*) be the number of trees of size *n* for a given node set. We define an algorithm, MEMORIZING-RANDOM-TREE-SEARCH, that uses Iba's algorithm as a subroutine to implement memorizing random search over the space of trees. This provides a reference algorithm that we can use to evaluate performance claims for enumeration on function trees.

Input: an oracle *O*, which takes a function tree *t* and answers *true* if the function tree is correct and *false* otherwise

Output: a valid tree *t* that solves the problem

```
$n \leftarrow 1$
$usedTrees \leftarrow \{\}$
for ever
  do
    $t \leftarrow$ GENERATE-RANDOM-TREE $n$
  while $t \in usedTrees$
  if $O(t)$ then return $t$
  if $|usedTrees| = trees(n)$ then
    $n \leftarrow n+1$
    $usedTrees \leftarrow \{\}$
  end if
end for
```

Algorithm 1. MEMORIZING-RANDOM-TREE-SEARCH.

With this algorithm, we can readily compute the number of fitness evaluations required to achieve a 99% success probability, as is typical for computational effort statistics. Let the number of trees of a given size be *n*, and let the number of successes at that size be *s*. Suppose that we perform *k* independent draws from this set of trees. We can compute the probability of achieving a success before *k* draws using (1), where $t_{success}$ is the index of the trial at first success. Here we have simplified the problem by assuming that $s << n$, so that we can treat the failures on successive trials as independent.

$$P\left(t_{success} \le k\right) \cong 1 - \left(\frac{(n-k)}{n}\right)^{s} \tag{1}$$

This expression can be solved for the unknown *s*, as in (2). We can then compute the number of trials to 99% success for a given tree size by substituting $P\left(t_{success} \le k\right) = 0.99$ in (2) and solving for *k*.

$$k \cong n\left(1 - \left(1 - P\left(t_{success} \le k\right)\right)^{1/s}\right) \tag{2}$$

Our final estimate for the 99% computational effort is then given by (3), where *m* is the smallest tree size at which any successes are available.

$$CE_{99} \cong trees(m)\left(1 - (0.01)^{1/s}\right) + \sum_{i=1}^{m-1} trees(i) \tag{3}$$

We can demonstrate this algorithm by considering the data of [3] for the Santa Fe trail problem. Figure 1 gives the number of trees and number of successes by tree size for the Santa Fe trail problem, as discovered by enumeration. With these data and using (3), we can compute the 99% computational effort for Santa Fe trail using MEMORIZING-RANDOM-TREE-SEARCH as 1,450,955 fitness evaluations. This differs from the exact value computed without the independence approximation by 1 fitness evaluation, and is conservative. It is also larger than the value of 450,000 quoted in [3], as we have included all the preliminary fitness evaluations in our count. Langdon and Poli considered only those trees of size 18, following the usual convention in GP for quoting computational effort statistics of ignoring preliminary work to find good parameter settings.

n	***trees*(*n*)**	***s*(*n*)**
≤ 7	5 043	0
8	20 412	0
9	95 256	0
10	516 132	0
11	2 554 416	12
12	13 712 490	48

Figure 1. Number of trees $trees(n)$ and successes $s(n)$ for different tree sizes *n*, for the standard problem for artificial ant on the Santa Fe Trail problem with 600 time steps.

2.2 Small Trees Analysis

We can improve on standard genetic programming by adding in subroutines derived from the analysis of successful small trees. To give a sense of this procedure, suppose that we enumerate all the trees of size 1, 2, 3, etc. Trees of size 1 are uninteresting, as they cannot be abstracted to a subroutine. For the artificial ant on the Santa Fe trail problem, there are no trees of size 2, so we first consider with trees of size 3. There are two high-performing trees of size 3, namely the two trees shown in Figure 2: they each have the best fitness of 78.

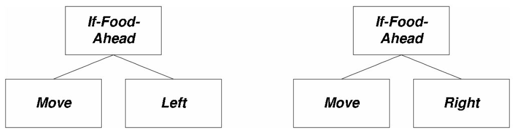

Figure 2. The two highest-performing trees of size 3 or lower for the Artificial Ant on the Santa Fe Trail problem.

We can look for commonalities in these two high-performing subtrees to determine a unique subtree. Fortunately, the common subtree is easily identified – the two subtrees differ only in a single node. Abstracting this subtree out, we get the tree of Figure 3.

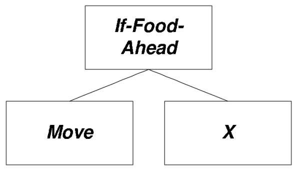

Figure 3. An abstraction of the two highest-performing trees of size 3 for the artificial ant on the Santa Fe trail problem. The "X" marks the node that will become a free parameter in the new subroutine.

This abstraction is, in fact, very close to a conventional subroutine. Suppose that we add the tree of Figure 3 back into the original problem as a new unary function node, If-Food-Ahead-Move(X). We take the common variation point as a free parameter of the new subroutine. If we then run MEMORIZING-RANDOM-TREE-SEARCH on the extended problem, we get the performance and trees shown in Figure 4.

n	***trees*(*n*)**	***s*(*n*)**
≤ 7	15 771	0
8	74 091	0
9	432 183	12
10	2 573 859	142
11	15 538 719	1 172

Figure 4. Number of trees $trees(n)$ and successes $s(n)$ for different tree sizes *n*, for the artificial ant on the Santa Fe Trail problem with 600 time steps using the additional unary function node If-Food-Ahead-Move(X) shown in Figure 3.

We can now compute the work to 99% success using the method of section 2.1. A quick substitution gives 227,602 fitness evaluations. To be fair, we must add in the fitness evaluations that we performed while considering all the trees of size 3 or smaller, which number 21 in total. The total computational effort is then 227,623 fitness evaluations: a factor of 6.37 easier than the normal computational effort.

2.3 Iterating the Small Trees Analysis

We can, of course, iterate this procedure. We should be careful in doing so, as we have now added a new node type to the function set of the artificial ant on the Santa Fe trail problem. As it is a new unary function, we are not going to gain any advantage by considering trees of size 2 or smaller, since the best of them will simply regenerate the work that we have already done. We should therefore begin our small tree scan at trees of size 3 and larger. The best trees by size for the augmented system are listed in Figure 5. Since the fitness of the trees of size 3 and 4 are no better than the best trees of size 2, we would not expect a savings by making subroutines from any of these trees. At size 5, we find 4 trees that share the size-optimal fitness of 65. It is these trees that we will consider further for subroutine making.

n	*trees*(*n*)	*best fitness*	*trees with best fitness*
2	3	78	2
3	21	78	4
4	84	78	16
5	435	65	4
6	2 343	51	1

Figure 5. Number of trees $trees(n)$, best fitness, and number of trees with the best fitness value for different tree sizes *n*, for the artificial ant on the Santa Fe Trail problem with 600 time steps using the additional unary function node If-Food-Ahead-Move(X) shown in Figure 3.

Looking at the 4 trees with exceptional fitness of size 5 for this new problem, shown in Figure 6, we see that they do not share any tree shape, much less any nodes in common.

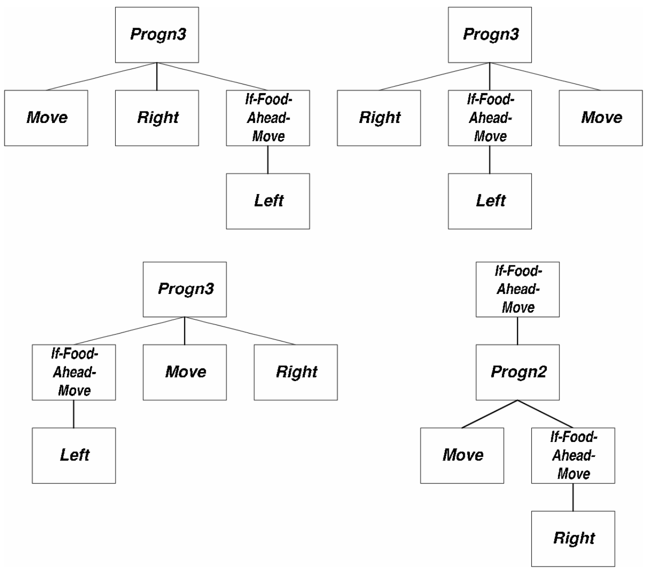

Figure 6. The four highest-performing trees of size 5 or lower on the Santa Fe trail problem, augmented with the added function If-Food-Ahead-Move(X) of Figure 3.

We may, for argument's sake, choose the first tree to generalize into a subroutine. Without any additional knowledge, let us take all three terminals and make them parameters of a new subroutine. Call this function If-Food-Ahead-Move-3(X, Y, Z); it is shown in Figure 7.

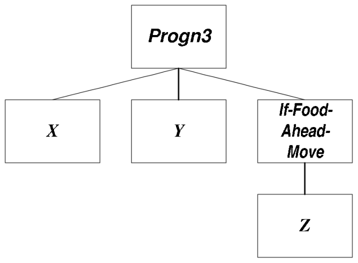

Figure 7. An abstraction of the first of the four highest-performing trees of size 5 for the Santa Fe trail problem, augmented with If-Food-Ahead-Move(X). "X", "Y" and "Z" mark nodes that become free parameters in the subroutine, If-Food-Ahead-Move-3(X, Y, Z).

As before, we add the tree of Figure 7 back into the revised problem as a new arity-3 function node. If we then run MEMORIZING-RANDOM-TREE-SEARCH on Santa Fe ant with both subroutines, we get the performance and trees shown in Figure 8.

n	*trees*(*n*)	*s*(*n*)
≤ 6	4 104	0
7	20 469	4
8	127 767	38
9	826 059	280

Figure 8. Number of trees $trees(n)$ and successes $s(n)$ for different tree sizes *n*, for the artificial ant on the Santa Fe Trail problem with 600 time steps using the additional unary function node If-Food-Ahead-Move(X) shown in Figure 3 and the arity-3 function node If-Food-Ahead-Move-3(X, Y, Z) of Figure 7.

We can now compute the work to 99% success for this augmented tree as before. We derive that 18,100 fitness evaluations are required for this version of the problem. As before, we must add in the fitness evaluations that we performed while considering all the trees of size 5 or smaller, which number 543 in total. We must also add in the 21 trees that we considered in generalizing the first subroutine. The total computational effort is then 18,664 fitness evaluations: a factor of 12.2 better than the previous, and 77.7 times better than the original computational effort.

The possibility remains that this is a fluke, that we were uncharacteristically lucky in choosing the first of the four trees to generalize in Figure 6. In Figure 9, we have illustrated key statistics from performing trial all-terminal subroutine generalizations from each of these four superior trees.

Tree	**Name**	m	$\sum_{i=1}^{m-1} trees(i)$	$trees(m)$	$s(m)$	CE_{99}
1	After	7	4 104	20 469	4	18 664
2	In	8	24 573	127 767	14	60 952
3	Before	8	24 573	127 767	8	81 055
4	Binary	8	32 286	171 615	35	54 008

Figure 9. Some statistics for solving Santa Fe Trail as in Figure 8, but with each of the four possible generalizations of the trees shown in Figure 6. The computational effort to 99% success, CE_{99}, includes all preliminary work required.

As Figure 9 illustrates, in each case the performance is better than that of the single-subroutine version, although we were a bit lucky in that we happened to choose the best candidate to generalize. The question arises: how can we automatically choose which tree from which to make a subroutine? One obvious technique would be to alternate randomly among all competing options, thus guaranteeing performance that is not too bad. Unfortunately, this technique has two disadvantages. First, since the mean value will be weighted by the poorest performing outliers, this may not be an efficient strategy. Second, we no longer have a single subroutine to generalize from, but rather a set of subroutines that are competing for success. If we wish to iterate this procedure, we will end up with combinatorially many sets of subroutines to choose from, which would be unwieldy at best.

Another technique would be to use the best fitnesses as a function of tree size to discriminate between the variants. This has the advantage of using information provided by the fitness function to guide our choice of subroutines to generalize. Here, we again perform all applicable generalizations, but we greedily choose the best variant among the alternatives available at a given size. If all the variants tie at a given size, we go to the next larger size. The process repeats until the tie is broken, or until we have exhausted a predetermined budget of fitness evaluations. In Figure 10, we show the best fitness at a given size, and the number of trees with that best fitness for all trees of sizes 2 through 5 for each of the function sets made reference to thus far. We skip trees of size 1 since no nullary functions were considered in this series.

n	***Normal***	***+ IFAM***	***+ IFAM + Var. 1***	***+ IFAM + Var. 2***	***+ IFAM + Var. 3***	***+ IFAM + Var. 4***
2		78 x2	78 x2	78 x2	78 x2	78 x2
3	78 x2	78 x4	78 x4	78 x4	78 x4	65 x1
4	86 x6	78 x16	65 x1	65 x1	65 x1	65 x3
5	78 x16	65 x4	51 x1	59 x2	51 x1	43 x1

Figure 10. Best fitness at a given size *n*, and the number of trees with this fitness, for each of the function set augmentations discussed thus far. In the Santa Fe trail problem, smaller scores are better, and 0 is a perfect score. The notation "*X* x*Y*" denotes a best score of *X* at the given size, with *Y* trees having this fitness. IFAM refers to the function If-Food-Ahead-Move(X); the four variants labelled "Var. 1", "Var. 2" and so on refer to the indexed variants of Figure 9. The subroutines corresponding to Variants 1, 2, and 3 have arity-3; Variant 4 is arity-2.

Considering Figure 10, this greedy heuristic would work as follows. When we add a new subroutine of arity *a*, the new subroutine has the possibility of improving the performance of all small trees of size $a+2$ and larger. Since a subroutine made from a tree with arity *a* can exactly duplicate the generating tree at size $a+1$, it will take a tree of size at least $a+2$ to *improve* on the performance. Therefore, we begin our comparison technique at trees of size $a+2$. Since Variant 4 has arity 2, it is the first tree to be compared at tree size 4. It would be unfair to not compare against the arity-3 trees of Variants 1 through 3, so we generate all trees of size 5 for all 4 variants. We also generate all trees of size 4 for Variant 4, since it may be better than the size 5 variants. This exhausts 2,499 fitness evaluations in total, and reveals Variant 4 as the lone winner: the best fitness for Variant 4 at tree size 5 is better than all alternatives.

One advantage of this algorithm is that we have already done the work required to get the next tree to generalize. We can then iterate the subroutine generalization step until we have exhausted our budget of fitness evaluations. For the Santa Fe ant problem, this results in one more subroutine generated from the single fitness 43 individual determined in the last step. This new subroutine is a new trinary function. Adding in this new subroutine gives the success data of Figure 11.

n	***trees*(*n*)**	***s*(*n*)**
≤ 6	6 705	0
7	37 209	13

Figure 11. Number of trees $trees(n)$ and successes $s(n)$ for different tree sizes *n*, for the artificial ant on the Santa Fe Trail problem with 600 time steps using the additional unary function node If-Food-Ahead-Move(X), the trinary function node If-Food-Ahead-Move-3(X, Y, Z), and the function node generated from the fitness 43 individual of Figure 10.

We can now compute the work to 99% success for this augmented tree as before. Naively, this version of the problem requires 17,804 fitness evaluations to get a solution with 99% confidence. As before, we must add in the fitness evaluations that we performed while generating each of the previous functions. We used $21+543$ evaluations to get the first two functions. The third function required 2,499 evaluations to discriminate among the choices; the discrimination duplicates the work required to choose the best function to generalize. Therefore, the total work to 99% success in this case is 20,867 fitness evaluations.

We can use the optimization described earlier as well to trim this number down. Specifically, for a new subroutine of arity *a*, there is no point in testing trees using this subroutine of size $a+1$ or smaller. While we made use of this trick for the third subroutine we generated, we did not use it for the first or second subroutines. This avoids testing the 3 trees of size 1 for the first subroutine, and the 3 trees of size 2 for the second subroutine. We also avoid performing 165 fitness evaluations for the trees of size 4 and below included in the counting of Figure 11. This gives us a grand total of 20,696 fitness evaluations for the Santa Fe ant problem.

2.4 Stopping Subroutine Generation

The question arises of when to stop adding new subroutines. As we alluded to earlier, we should determine beforehand a threshold for the number of evaluations that we wish to use to generate subroutines. This establishes a "budget" of fitness evaluations, within which we are free to do as we please. We then carefully monitor this budget, and abort subroutine generation when the number of fitness evaluations required goes over our predetermined limit. Fortunately, we are aided in this effort by the existence of a simple and efficient algorithm to count the number of trees of a given size and node choice [7]. This lets us precompute how much work will be required to test all the competing small trees, so we can easily test whether it is worthwhile to attempt subroutine generalization in a given case.

As an example of this procedure, consider the next step in the previous generalization sequence. There are 4 trees with fitness 37 of size 6 using the three subroutines found earlier. These 4 trees would be candidates for subroutine generalization; they all have arity-4 and do not share the same geometry, so are incommensurable. Since $a = 4$, we need to consider trees of size 6 and larger. A quick computation shows that there will be 6,078 trees of size 6 for each of the 4 variants to consider. This will require an additional 24,312 evaluations to perform, on top of the 9,600 evaluations performed already by that point. If we allocate a budget of 10,000 fitness evaluations for subroutine generation, then we would cut off the subroutine generation procedure at that point, and proceed to test trees of size 7 with the three subroutines already discovered.

In general, we will always carry out a side computation first to see if it will cost too much to try to generate subroutines from high-scoring subtrees. We may find ourselves upping this estimate if no better subtrees to generalize are found during the procedure, as with the If-Food-Ahead-Move subroutine described at the start of section 2.3. A list of the work estimates for the Santa Fe ant problem is given in Figure 12.

Subroutines	*n*	*Evaluations so far*	*Evaluations to test subroutines*
(none)	1	3	
	3	21	≥ 42
+ IFAM	3	42	≥ 126
	4	126	≥ 561
	5	561	≥ 3 060
+ Variant 4	5	3 060	≥ 3 927
+ Subroutine 3	5	3 927	≥ 9 600
	6	9 600	≥ 33 912

Figure 12. The sequence of subroutine-generalization steps carried out in the present work. For each step, we have shown the subroutines added, the tree size *n* under consideration, the number of evaluations performed thus far, and the estimate of the number of evaluations that will be required to generate a new subroutine.

As Figure 12 shows, our choice of 10,000 fitness evaluations is not too arbitrary. We would have derived the same result if the cutoff were anywhere between 3,927 and 33,911 fitness evaluations. In practice, an efficient algorithm might consider progressively more fitness evaluations to devote to subroutine-generation as success is lacking. For instance, we might want to spend 25% of our fitness evaluations on generating subroutines, and the rest on actually solving the problem. We do not pursue such an algorithm any further in this paper.

2.5 Subroutines With GP

In an ideal situation, the subroutines generated by the techniques described herein would be useful in standard genetic programming. To test this hypothesis, we used Luke's ECJ [8] to determine the computational effort to 99% success for the Santa Fe Trail problem in a control and three experimental conditions. We used standard GP as a control. For the experimental condition, we used GP augmented with one or two automatically-generated subroutines, as derived by the methods given above. We used the reference implementation of the artificial ant on the Santa Fe trail, which uses tournament selection of size 7 and evaluates to 600 fitness evaluations. To reduce the running time and thereby increase the number of runs performed, we limit tree sizes to 50 nodes. We performed at least 10,000 runs for each setting, or enough runs to derive a computational effort accurate to within 5% of the true value, 95% of the time. We performed runs with the population size and generation number chosen to be powers of 2 so as to get nearly optimal settings for the computational effort in each case. The results are shown in Figure 13.

Condition	*Random Search*	*M*	*G*	*with GP*
Standard GP	1 450 954	1 100	17	440 000
+ 1 subroutine	227 623	1 000	14	200 000
+ 2 subroutines	*53 670 (avg.)*	250	1	40 000
+ 3 subroutines	*20 696*			

Figure 13. Computational effort to 99% success for MEMORIZING-RANDOM-TREE-SEARCH and genetic programming for the 4 function sets of this paper on the Santa Fe Trail problem. For the genetic programming runs, *M* is the population size and *G* the optimal generation number for the GP. The GP data are accurate to 5%; the random search data are accurate within 1 evaluation.

From the data of Figure 13, we can see that the same subroutines that improve performance on MEMORIZING-RANDOM-TREE-SEARCH also help genetic programming. The genetic programming numbers of Figure 13 are computed in the usual way: work required to find the optimal parameter settings is ignored. This differs from the situation with memorizing random search, where all evaluations are counted. The data of Figure 13 neglect an initial 42 fitness evaluations required for GP with 1 subroutine, and an initial 3,060 evaluations for GP with 2 subroutines – for fairness, they should be included as well in the tallies above.

3. RESULTS

We can compare the results derived here against published values for the computational effort of artificial ant on the Santa Fe trail. The best collection of published results on the Santa Fe Trail problem is in [3]. We derive our Figure 14 from Table 3 in that work, where we show the best computational efforts for each treatment in the published literature.

Algorithm	*Ref.*	*Effort*
Koza GP	[2]	450 000
Subtree Mutation	[9]	426 000
Size Limited EP	[4]	136 000
Random trees: size 18	[3]	450 000
Simulated Annealing	[3]	435 000
PDGP	[3]	336 000
Strict Hill Climbing	[3]	186 000
This Work, Standard GP		*440 000*
This Work, Random Search + Subroutines		*20 696*
This Work, GP + Subroutines		*43 000*

Figure 14. A comparison of various published results for the computational effort to 99% success on the artificial ant on the Santa Fe trail problem with the results of this paper. The referenced results are quoted without error bars; for typical run counts used in GP, 95% confidence intervals cover about a factor of 2. Data presented here are accurate to at least 5%.

4. DISCUSSION

The results of this work clearly demonstrate that the methods of this paper are successful. The computational effort to 99% success for MEMORIZING-RANDOM-TREE-SEARCH is a factor of 7 less than any published result for this problem. In addition, Christensen and Oppacher [10] showed that Koza's computational effort has some statistical troubles, especially when fewer than 100 runs are performed. In particular, they showed that reported computational effort numbers tend to underestimate the true values when few runs are performed. We heed their advice in the experiments in section 3, where we perform 10,000 runs of each treatment to avoid statistical issues with the computational effort statistic. This underestimation may befall some of the published results quoted in Figure 14, suggesting that the true values of computational effort for the listed methods are more likely to lie above the quoted values than below them. Since Langdon and Poli published the raw data for their size 18 random search data in Figure 3 of [3], we can conclude that the quoted value of 450,000 evaluations has a 95% confidence interval on the relative error of about 20%.

We have not, of course, considered the generalizability of this procedure to other problems. The artificial ant on the Santa Fe trail is an interesting problem, with a non-zero probability of a perfect solution under typical GP conditions. We might expect that the methods described here would be useful for related problems where an exact success is possible. However, it is not clear at all that these techniques will be beneficial for problems where the probability of exact success is vanishing. Examples of the latter would include problems with continuous-valued fitness functions such as symbolic regression [2]. Indeed, problems like symbolic regression that have ephemeral random constants (ERCs) [2] may not benefit from the methods of this paper as the terminal set is effectively infinite in principle. However, the methods presented herein can be used as is once a set of ERCs are assigned.

The procedure detailed here effects a compression of the GP function tree; in this way, it acts a little like a data compression algorithm. It differs from constructive GP systems in that the data to be compressed, the successful subroutines, are derived from promising small tree candidates, not from genetic information within the individual itself. The idea of manufacturing new function nodes from successful subtrees can be viewed as biasing the search space towards trees that contain subunits that are independently successful. Indeed, this is what genetic programming itself tries to do, and so it perhaps unsurprising that the technique presented here is successful. What may be surprising is that it so greatly outperforms standard genetic programming. This suggests that on the Santa Fe trail problem at least, the kind of local search presented here is a very productive addition to genetic programming. Indeed, we have also published the best known result for the Santa Fe trail problem using a variant of genetic programming, beating the previous record by a factor of roughly 3, namely 43,000 evaluations to Chellapilla's 136,000.

5. CONCLUSIONS

In this paper, we present a method for automatically generating useful subroutines by systematically considering all small trees. Iterating this procedure allows the automatic generation of several subroutines, along with a reasonable criterion for terminating the search for subroutines. The method presented here achieves a computational efficiency on the artificial ant on the Santa Fe trail problem that is a factor of 7 less than any published result for this problem, and is a factor of roughly 70 less than memorizing random search. We also show that the subroutines thus conceived can be useful in a conventional genetic programming run. The use of these subroutines reduces the work required to solve the artificial ant on the Santa Fe trail problem by a factor of 3 over any previously published result. We recommend this subroutine-generating algorithm as a preliminary stage for any problem with a finite function and terminal set.

6. REFERENCES

[1] Koza, J.R. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, USA. 1994.

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

[3] Langdon, W.B., and Poli R., Why Ants Are Hard. *Genetic Programming 1998: Proceedings of the Third Annual Conference*. 193-201. Morgan Kaufmann, Madison, USA. 1998.

[4] Chellapilla, K. Evolutionary Programming with Tree Mutations: Evolving Computer Programs Without Crossover. In Koza, J.R. et al., eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*. Morgan Kaufmann. 1997.

[5] Luke, S. and Panait, L. A Survey and Comparison of Tree Generation Algorithms. In Spector, L., Goodman, E.D., Wu, A., Langdon, W.B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M.H., Burke, E., eds. *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*. 81-88. Morgan Kaufmann, San Francisco, USA. 2001.

[6] Iba, H. Random Tree Generation for Genetic Programming. In Voigt H.-M., Ebeling W., Rechenberg I., Schwefel, eds. *Parallel Problem Solving From Nature IV, Proceedings of the International Conference on Evolutionary Computation,* LNCS 1141. 144-153. Springer Verlag, Berlin, Germany. 1996.

[7] Christensen S., *Towards Scalable Genetic Programming*, Ph.D. Thesis. 122-132. Ottawa-Carleton Institute for Computer Science, Ottawa, Canada. 2007.

[8] ECJ: A Java-based Evolutionary Computation and Genetic Programming System. Available at http://cs.gmu.edu/~eclab/projects/ecj/. 2007.

[9] Langdon W.B., and Poli, R. Fitness Causes Bloat: Mutation. In Banzhaf W., et al. eds. *Proceedings of the First European Workshop on Genetic Programming*. Springer-Verlag. 1998.

[10] Christensen, S., and Oppacher, F. An Analysis of Koza's Computational Effort Statistic for Genetic Programming. In Foster, J.A., Lutton, E., Miller, J.F., Ryan, C., Tettamanzi, A., eds. *Genetic Programming, 5th European Conference, EuroGP 2002*. LNCS 2278. 182-191. Springer-Verlag, Heidelberg, Germany. 2002.

[11] Wolpert, D.H. and Macready, W.G. *No Free Lunch Theorems for Search*. Technical report SFI-TR-95-010. Santa Fe Institute, USA. 1995.