Hill Climbing Beats Genetic Search on a Boolean Circuit Synthesis Problem of Koza's

Kevin J. Lang
NEC Research Institute
Princeton, NJ 08542
kevin@research.nj.nec.com

Abstract

An experiment described in chapter 9 of the book "Genetic Programming" shows that the method is more efficient than random-generate-and-test on a boolean circuit synthesis task. Here we show that hill climbing is more efficient than genetic programming on this problem. It is interesting to note that our improved results were obtained by mating the current best hypothesis with completely random S-expressions, rather than with members of a high-fitness population. Perhaps, for this task, fragments of high fitness individuals have no special value when transplanted into other individuals.

1 Introduction

The book "Genetic Programming" demonstrates the method's generality by exhibiting solutions to a variety of problems. However, it provides little evidence for believing that GP is better than the many existing algorithms for general learning and optimization, much less than the specialized algorithms that exist for many of the various problems. Genetic programming has attracted a large following anyway, probably because it comes with an appealing story about how the power of evolution is being harnessed by the technique. A crucial piece of this story is the idea that the pool of highly fit individuals selected from earlier generations constitutes a valuable genetic resource that facilitates the creation of even more fit individuals in the future.

This paper describes an experiment that casts doubt not only on the relative efficiency of genetic programming, but also on the validity of the story about the value of the high-fitness gene pool. The testbed for the experiment is the boolean circuit synthesis task described in chapter 9 of "Genetic Programming". This task was selected by the book's author as the ground on which to battle an earlier batch of critics who suspected that genetic programming was nothing more than a disguised version of the random-generate-and-test algorithm (henceforth called RGAT). Extensive testing on the circuit synthesis task showed that genetic programming is more efficient than RGAT, and that the advantage enjoyed by genetic programming increases on target functions that are more difficult to learn.

We observed that the margin of victory over RGAT was small in this experiment, and were thus motivated to compare GP with the next weakest search method, hill climbing. We found hill climbing to be more efficient than GP, with the advantage increasing to about a factor of 30:1 on the hardest problem instances. Our version of the hill climbing algorithm is closely related to GP. The main difference is that our algorithm generates new hypotheses by crossing the current champion with totally random individuals, while GP generates them by crossing pairs of individuals drawn from a high-fitness pool. Evidently, the high-fitness pool is a worse than random source of components for building improved individuals.

2 Genetic Programming

Genetic Programming is a learning method that searches a hypothesis space of tree-structured circuits (represented as Lisp S-expressions), looking for a circuit that computes a given target function. Each circuit in the space accepts input values through its leaf nodes, processes them using basis functions associated with its interior nodes, and emits an answer from its root node. The hypothesis space is searched using a genetic algorithm: First, an initial population is drawn from some probability distribution over the space. Then successive generations of circuits are created by selecting the best trees[1] from preceding gen-

[1] *i.e.* the circuits that come closest to computing the desired function.

erations. The number of copies that are made of a selected tree is an increasing function of the tree's fitness. In addition, some of the selected trees are fed into a crossover operation to create new trees that are mixtures of their parents rather than strict copies. The crossover operation consists of selecting a random node in each of the two parents, and then swapping the subtrees rooted at those two nodes. The overall procedure for constructing a new generation is somewhat complicated, and is controlled by several adjustable parameters. The reader is referred to "Genetic Programming" for the parameter settings employed during the GP vs RGAT comparison.

3 GP vs RGAT

In this section we summarize the experimental setup and results reported in chapter 9 of "Genetic Programming". The benchmark problem was the task of finding logic circuits to compute the various boolean functions of 3 inputs. There are 256 such functions, but a set of 80 representatives covered the variety of the full set. The hypothesis space for the experiment was the set of tree-structured circuits in which a leaf accesses one of the 3 input values and each internal node computes one of the functions (AND, OR, NAND, NOR).

The first search method was RGAT. Random candidate circuits were drawn from the uniform distribution over 20-gate circuits. Each candidate was tested to determine whether it computed the desired function. The figure of merit for this technique was the reciprocal of the probability of randomly drawing a circuit which computes the target function, or in other words, the ratio of the number of candidates that were examined to the number of solutions that were found.

The second method was genetic programming. An initial population of 50 candidates was drawn from the uniform distribution over 20-gate circuits. Then, somewhere between 0 and 24 additional generations were constructed using the genetic algorithm outlined in the previous section of this paper. The search was terminated whenever a generation contained a solution, or when the limit of 25 generations was exceeded without finding a solution. Many trials of this procedure were performed, and the figure of merit was the ratio of candidates to solutions, as measured during the complete set of trials.

The results of this experiment are plotted in figure 1. Each of the points marked with a "+" in the figure shows the performance of GP and RGAT on one of the 80 target functions. The x coordinate of each point represents the ratio of candidates to solutions for GP, while the y coordinate represents the same information for RGAT. The dotted line running diagonally through the middle of the figure represents equal performance for the two algorithms. Since most of the points lie above this line, RGAT was less efficient than GP.

4 Hill Climbing

We applied the following hill-climbing algorithm to the circuit synthesis task: remembering the best circuit ever seen, repeatedly generate new circuits and compare them with the best one. A candidate becomes the new best circuit if it is at least as good as the old one. On alternate steps the candidate is either a random circuit or a hybrid circuit obtained by crossing the best circuit with a random circuit.

In our experiment, all random sampling was from the same distribution that was employed in the earlier experiment for RGAT, namely the uniform distribution over 20-gate circuits.[2] Hybrid circuits were constructed with the same cross-over operation that had been employed by GP.

We ran 100 trials on each of the 80 target functions. Each trial was halted when a solution was found, or when 1250 candidates had been examined without finding a solution.[3] The figure of merit was the ratio of candidates to solutions during the complete set of trials for a given target function.

Our results for the various target functions are plotted as small diamonds in figure 1. The x coordinate of a point represents the ratio of candidates to solutions for GP, while the y coordinate represents the same ratio for hill-climbing. Most of the points lie below the dotted line, thus showing that hill climbing is more efficient than genetic programming on this task. The superiority of hill-climbing appears to increase with target function difficulty. The mean improvement on the five hardest targets was by a factor of 33.

5 Interpretation and Speculation

Our hill climbing algorithm is closely related to the RGAT and GP algorithms that were employed in the earlier experiment. Fully half of our candidate circuits were random, drawn from the same distribution that yielded poor performance for RGAT. While this seems like a waste of resources, we needed to have some mechanism for escaping from the local optima that would have resulted from keeping only one good circuit around at a time. We believe that these aspects

[2] Sampling from this distribution is a bit tricky. We verified that our implementation was the same as in the earlier experiment by generating millions of circuits and comparing the resulting histogram with table 9.3 in the book.

[3] This rule corresponds to the 25-generation limit in the earlier experiment.

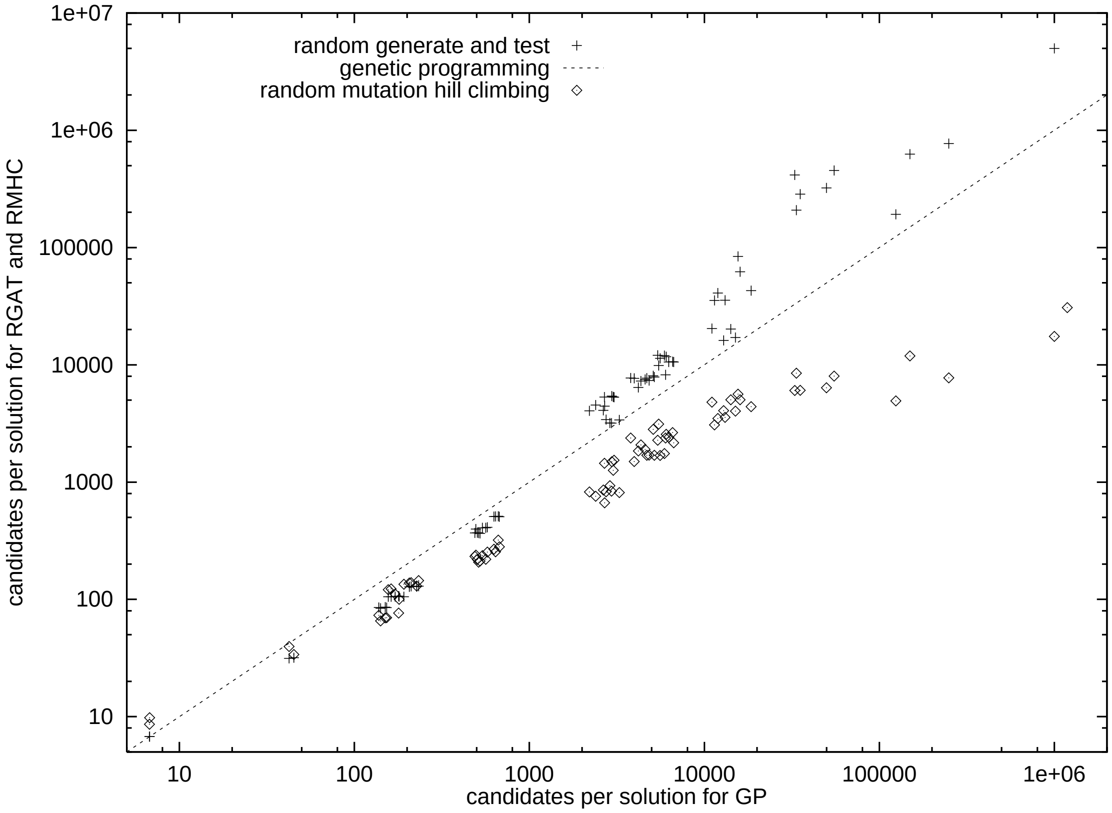

Figure 1: Each point in this figure shows how the efficiency of an algorithm compared with that of GP on one of the 80 target functions. Points lying above the dotted line indicate that more candidates had to be examined by the alternative method than by GP, while points lying below the dotted line indicate that fewer candidates were required.

of our algorithm are a disadvantage when compared to GP.

Our remaining candidates were generated by crossing our one good circuit with completely random circuits, again drawn from the distribution that yielded poor performance for RGAT. By comparison, GP generated new candidates by crossing pairs of good circuits from the previous generation. Since we found solutions much faster than GP did, apparently the high-fitness population maintained by GP was a worse than random source of components for building improved individuals. How could that be? If fragments of high fitness individuals have no special value when transplanted into other individuals, then relying on a small high-fitness population as a source of new genes confers no advantage, but has the disadvantage of restricting the variety of new individuals.

We note that the earlier study included a limited experiment on the effect of larger population sizes on the learning of target function 150. It was found that every increase in the population size yielded a performance increase. The best reported result was a candidate to solution ratio of 20285 using a population size of 1000. Our candidate to solution ratio for that function was 17499. In a sense, our population size is infinite, since the random tree generator provides an effectively unlimited number of new trees to mate with. Clearly these random trees have no special value as a gene source. We suspect that the same thing is true for the trees in the high-fitness pool employed by GP.

6 References

John Koza, "Genetic Programming", MIT Press, 1992, pp. 205-236.

target	RGAT	GP	RMHC1	RMHC2
0	6.76	6.75	9.78	9.22
255	6.76	6.74	8.61	9.14
85	31.4	42.3	39.63	35.6
240	31.8	45.1	33.89	28.7
119	84.0	141.0	65.57	53.6
192	85.1	153.0	69.81	64.7
17	85.2	138.0	73.14	59.6
252	85.8	150.0	69.58	70.0
1	105.3	156.0	120.81	110.9
128	105.3	162.0	122.98	112.6
127	105.4	192.0	134.53	96.6
245	105.5	180.0	100.21	74.1
254	105.8	171.0	111.19	122.3
80	106.5	179.0	76.38	72.0
16	127.8	206.0	138.08	129.3
253	128.4	227.0	129.82	98.0
64	129.1	211.0	138.93	106.9
247	129.4	233.0	144.53	100.8
200	365.5	522.0	212.13	234.1
87	367.9	513.0	207.79	192.4
21	368.8	488.0	232.6	191.0
236	370.4	504.0	219.12	197.0
213	397.2	495.0	238.13	193.7
205	407.6	539.0	235.99	227.5
84	408.0	564.0	219.84	253.1
112	412.1	575.0	253.18	246.5
81	507.4	674.0	280.91	264.8
117	508.5	627.0	267.79	256.7
244	509.8	641.0	254.57	266.5
162	511.7	664.0	319.94	298.2
58	3180.7	2946.0	836.52	813.9
197	3188.8	2888.0	933.15	667.9
53	3393.3	3264.0	813.05	951.9
216	3409.5	2746.0	826.45	747.2
23	4050.2	2204.0	825.0	717.0
232	4093.3	2645.0	855.90	900.0
102	4438.5	2694.0	666.92	527.4
153	4543.4	2391.0	756.99	758.8
20	5299.4	3051.0	1533.68	1116.7
235	5313.5	2689.0	1447.88	1070.2

target	RGAT	GP	RMHC1	RMHC2
159	5324.8	3015.0	1259.98	1228.7
96	5417.1	2959.0	1492.01	1247.8
9	6418.5	4201.0	1840.36	1739.5
212	7272.7	4338.0	2073.46	1539.2
193	7331.4	4836.0	1697.96	2037.4
190	7513.1	4585.0	1902.71	1887.7
77	7674.6	3970.0	1498.62	1433.1
111	7686.4	4694.0	1695.62	1993.7
144	7716.0	3796.0	2376.36	1368.8
122	7861.6	5175.0	1695.55	1861.1
133	8064.5	5105.0	2812.43	1518.5
118	8216.9	6006.0	2377.10	1809.4
164	9861.9	5482.0	3120.36	2372.6
67	10559.7	6683.0	2160.30	2201.5
91	10570.8	6263.0	2404.92	2000.7
230	10649.6	6581.0	2642.97	2541.9
229	11350.7	5586.0	1688.42	1271.8
98	11737.1	6057.0	2554.46	1588.8
26	11990.4	5910.0	1752.00	1474.8
157	12048.2	5410.0	2277.90	1462.0
126	16155.1	12877.0	4065.92	3370.8
129	17094.0	15068.0	4030.03	2741.3
189	20202.0	14176.0	5035.04	4945.0
24	20366.6	11062.0	4811.43	3904.8
149	35461.0	11432.0	3075.32	3965.6
30	35587.2	13140.0	3560.86	3371.5
106	40983.6	11950.0	3491.06	2846.0
169	42918.5	18510.0	4398.76	3161.8
101	62111.8	16013.0	5038.31	6240.4
154	84033.6	15578.0	5620.00	6763.5
22	192307.7	124225.0	4923.91	4248.1
233	208333.3	33590.0	8481.92	6471.3
104	285714.3	35335.0	6057.57	4686.8
151	322580.6	49825.0	6371.0	4023.7
146	416666.7	32900.0	6050.36	6096.3
97	454545.5	55200.0	8023.6	8567.4
107	625000.0	149520.0	11902.6	5673.8
158	769230.8	249350.0	7737.8	6865.9
150	5000000.0	999750.0	17499.0	12053.6
105	$> 1.0E7$	1187225.0	30799.0	20443.2

Figure 2: This table summarizes the performance of RGAT, GP, and random mutation hill climbing on 80 instances of the boolean circuit synthesis problem. The target numbers are explained in chapter 9 of the book "Genetic Programming". The remaining columns show the number of candidates that were examined per solution by each of the algorithms. The RGAT and GP values were taken from tables 9.3 and 9.4 in the book. RMHC1 is the algorithm described in this paper. RMHC2 used a different probability distribution over possible mutations.