Hybridized Crossover-Based Search Techniques for Program Discovery

Una-May O'Reilly
Franz Oppacher

SFI WORKING PAPER: 1995-02-007

Hybridized Crossover-Based Search Techniques for Program Discovery

Una-May O'Reilly
Santa Fe Institute
unamay@santafe.edu

Franz Oppacher
Carleton University
oppacher@scs.carleton.ca

Abstract

In this paper we address the problem of program discovery as defined by Genetic Programming [10]. We have two major results: First, by combining a hierarchical crossover operator with two traditional single point search algorithms: Simulated Annealing and Stochastic Iterated Hill Climbing, we have solved some problems with fewer fitness evaluations and a greater probability of a success than Genetic Programming. Second, we have managed to enhance Genetic Programming by hybridizing it with the simple scheme of hill climbing from a few individuals, at a fixed interval of generations. The new hill climbing component has two options for generating candidate solutions: mutation or crossover. When it uses crossover, mates are either randomly created, randomly drawn from the population at large, or drawn from a pool of fittest individuals.

1 Introduction

An important question to the Genetic Programming (GP) [10] community is whether the evolution-based approach to program discovery, as espoused by GP, is generally superior to other adaptive search techniques. Or, because no single approach works best in all situations, it is important to learn the salient characteristics of program discovery fitness landscapes that make them amenable to a particular adaptive search algorithm. Another obvious goal is to capitalize upon the strategies of other adaptive search methods by incorporating them into an enhanced version of GP that outperforms the canonical GP.

We recently [11] designed a mutation operator, HVL-Mutate, that can be used with a variable length hierarchical representation. HVL-Mutate changes a "tree" via shrinking, growth or internal substitution while preserving syntactic correctness. We reported that mutation-based adaptive search with Hill Climbing or Simulated Annealing [1], using HVL-Mutate, can accomplish program discovery tasks. Furthermore, sometimes one of these single point based algorithms required fewer fitness evaluations to find a correct solution or found a solution more reliably (i.e. averaged over an ensemble of runs).

While that work sought to bring the program discovery problem into the realm of traditional search, this paper seeks to exploit and explore the nature of the genetic-based crossover operator. Starting from the idea of Crossover Hill Climbing, "$XOHC$", which Terry Jones [5, 6, 7] employed upon the fixed length binary string representation of GAs, it was simple and straightforward to implement a similar

$XOHC$ algorithm for hierarchical variable length representation by exchanging GA crossover for GP crossover. We describe this algorithm in Section 2.1 of this paper. On the "Block Stacking" and "Boolean Multiplexer" problems $XOHC$ outperformed all other search algorithms we have tried to date.

We next combined Crossover with Simulated Annealing, "$XOSA$". Based upon the successes of HVL-Mutate Simulated Annealing and Crossover Hill Climbing we conjectured favorable results. Rather puzzlingly, this algorithm did not perform as well as expected. It is described in Section 2.2 and its performance on each problem is provided in Section 4.

Finally, the results of Crossover Hill Climbing were sufficiently encouraging to suggest adding a hill climbing component to GP. Currently GP is a population-based technique that provides a means of quickly focusing search on a fit area of the search space. This is the recognized effect of the algorithm as, over time, the population becomes more homogeneous due to selection based upon fitness and the combinative effect of crossover. Mutation is usually set to a background rate and plays only a minor role in terms of exploitative search (it ensures no premature allele loss). However, once a standard GA or GP algorithm has found fit areas of the search space, it searches over only a small fraction of the neighbourhood around each search point. It must derive its power from integrating multiple single neighbourhood explorations in parallel over successive generations of a population. This "many points, few neighbours" strategy is in direct contrast to a hill climber or simulated annealer which potentially focuses effort on a greater fraction of the search neighbourhood of one point but only around one point at a time. This strategy might be called "few points, many neighbours". The two strategies could clearly be complementary [2, 3, 8, 4] with the GA component serving to "zero in" on regions of high fitness and the hill climbing component serving to thoroughly explore the regions the GA has found.

Therefore, our conjecture was that GP plus a Crossover Hill Climbing component might be profitable on our suite of problems. In Section 2.3 we describe the details of the "*GP+XOHC*" algorithm. The conjecture was borne out by experimentation. In particular, on the 11-bit Boolean Multiplexer problem, we were encouraged by an improved probability of success and a decreased requirement on fitness evaluations. We also implemented GP plus mutation-based hill climbing, "*GP+MU-HC*", which is also described in Section 2.3. This permits the comparison of the hybrid with a mutation based hill climber (using HVL-Mutate) to a crossover-based one. The obvious qualitative difference is that mutation introduces totally unselected genetic material while the crossover operator (if it draws a mate from the population at large or from the pool of fittest individuals) replaces swapped out genetic material with material that has undergone selection by surviving through GP's simulated process of evolution. This is a crucial distinction that reflects upon the efficacy of the hybrid or standard GP. If, on a given problem, random material proves as useful as duly evolved and selected material, single-point search algorithms such as Simulated Annealing and Hill Climbing may be a superior alternative to GP.

Section 2 summarizes the various search methods used. Section 3 describes the testsuite and experimental procedure. Section 4 of the paper covers the experimental results and discussion. Section 5 is the conclusion and plans for future work.

2 New Crossover-Based Search Algorithms

It is worthwhile to consider the dynamics of GP Crossover with respect to its range of search over the search space (i.e. its *neighbourhood*), given two mates. The *neighbourhood* of two mates involved in crossover is the set of all offspring that can possibly be derived from crossing over the two mates and that are different from them. The term *neighbour* is synonymous with offspring.

In GP Crossover, the maximum size of the neighbourhood is the product of crossover points in the recipient and the donor.[1] The actual size depends upon the sizes of the specific parents involved in the crossover and upon the redundancy of offspring (i.e. duplicate offspring are not counted in the size of a neighbourhood).

There is significant difference between the size of the crossover neighbourhoods in GP Crossover and in standard GA crossover. In GAs that use a fixed length representation, the maximum size of the GA Crossover neighbourhood is $2l$ where l is the length of the representation. The actual size of the neighbourhood is $2l$ less the number of duplicates. The fixed position representation of GAs implies that, if each mate has the same value at a given position, all offspring will only have that value at that position. Let us term this incident "allele redundancy". In a binary representation, the probability of allele redundancy at a bit position in two independent strings is 50% and this considerably constrains the neighbourhood size. With an alphabet of higher cardinality the probability of allele redundancy is less but nonetheless when the two mates are equal, the number of duplicates equals $2l$ and the neighbourhood size equals zero.

In GP crossover, there are two reasons to expect less redundancy among the neighbours of two mates. First, GP has a non-binary alphabet which reduces the probability of the recipient and donor containing an identical subtree. Second, because there is no fixed positioning in the representation, any primitive(s) in the donor can be placed anywhere in the recipient and thus provide another offspring. For example, consider two duplicate 2 node S-Expressions with a distinct root and child. This is akin to the case of a binary alphabet and identical mates. In GP there are 4 possible crossovers and while two of these produce duplicates, the remaining 2 produce original trees. As another example, consider two duplicate 3 node S-Expressions where the root has 2 children and each node is a distinct primitive. There are nine offspring in the crossover of the two mates but only three of the nine produce a duplicate.

In summary, with GP Crossover, a crossover neighbourhood is likely to be larger than that resulting from GA crossover because its maximum size is the product of both mates' sizes rather than $2l$ and because GP Crossover is likely to generate fewer

[1]In GP crossover the number of crossover points equals the number of primitives in the S-Expression (or nodes in the tree) so the maximum size is equals the product of mates' sizes.

duplicates than GA crossover.

2.1 XOHC Algorithm

Crossover hill climbing was first described by Terry Jones [5, 7]. Recall that hill climbing is a search algorithm that commences from a single solution point.[2] At each step a candidate solution is generated using a "move operator" of some sort. The algorithm simply moves the search from the current solution to a candidate solution if the candidate has better or equal[3] fitness. A parameter controls how many moves will be tried from the current point before a random one replaces it.

The basic idea of crossover hill climbing is to use hill climbing as the move acceptance criterion of the search and use crossover as the "move-operator". In this case, the candidate solution is, therefore, generated from one "current" solution and a random mate. Our version of GP crossover retains the essential spirit of GP crossover in [10] but is a simple two-parent to one-child function rather than a two-parent to two-children version. One parent is the "donor" and the other the "recipient". A randomly chosen subtree is copied from the donor and replaces a subtree randomly removed from the recipient.

The algorithm always maintains the fittest-overall-solution (i.e. the fittest point of all points examined) and a current-solution. At the outset, a mate for the current-solution is randomly generated. For some number of attempts, "*mate-xo-tries-limit*", offspring are generated via crossover from the pair of current solution and mate. If an attempt yields an offspring that is accepted, the offspring replaces the current solution and the process repeats with the number of crossover attempts reset to zero. If the number of crossover attempts reaches *mate-xo-tries-limit* without an offspring being accepted, a new mate is chosen for the current-solution. The number of times the current-solution is used, "*xo-tries-limit*", is also a parameter of the algorithm. After *xo-tries-limit* crossovers, the current-solution is discarded and a new one randomly generated. After a fixed number of fitness evaluations or when a perfect[4] solution is found the algorithm terminates and returns the fitness of the fittest-overall-solution.

We experimented with values for both parameters of this algorithm. Since a mate is randomly generated, the algorithm was not very sensitive to the value of *mate-xo-tries-limit*. However, *xo-tries-limit* is integral to the algorithm because it sets a limit for crossover attempts after which the search moves *randomly* elsewhere. If its value is set too low, the search may not find a fitter candidate even though one exists in the neighbourhood. If it is set too high, the search may be trapped in a local optimum. We used values equal to the most successful "*max-mutation*" [5] values in previously conducted HVL-Mutate Hill Climbing experiments [11] and found them satisfactory.

[2]We use the terms "solution" and "point"synonymously.

[3]Acceptance of equal fitness candidate solutions is optional in hill climbing. We have chosen to use it.

[4]Perfect means that a program scores the maximum fitness and successfully solves each test case.

[5]This parameter controls how many mutations of the current solution are tried before it is replaced by a random solution

2.2 XOSA Algorithm

The standard version of Simulated Annealing, "SA", [1] is based upon search which traverses from one solution point to another. A mutation of the current point is accepted and made the current point depending upon two criteria: the difference in fitness between it and the current point, and, upon the current temperature, T, of the SA system. A mutant is accepted outright if it is equal or superior in fitness to the current point. Otherwise, it is accepted with a probability that decreases with the temperature of the system and depending upon the fitness difference according to a Boltzmann distribution. The temperature in a SA system is cooled in discrete steps and if at each temperature the system reaches a simulated thermal equilibrium, convergence to a global optimum is guaranteed. In practice, such a cooling schedule is extremely slow but much faster approximations achieve good performance.

The SA component of our $XOSA$ algorithm is basic. It uses normalized fitnesses. The cooling schedule is determined by the *a priori* fixed initial (1.5) and final temperatures and upon the maximum number of evaluations.[6] The starting temperature is always decreased to the final temperature over the given number of evaluations according to an exponential cooling schedule.

The $XOSA$ algorithm uses GP Crossover to generate candidate points from two "current" solutions and SA to decide whether a candidate should be accepted. It has one non-standard parameter: *xo-tries-limit*. Every *xo-tries-limit* crossovers, the weakest current solution is replaced with a random program. This ensures sufficient novelty. The SA component can be viewed as a predicate, SA-ACCEPT, of 3 parameters. The formal parameters are: *fitness-of-current-state*, *fitness-of-candidate-state*, and *temperature*.[7] The predicate uses the current temperature, a calculated fitness differential and a random number generator to indicate whether the candidate state of the system should be accepted.

We designed three different versions of $XOSA$: *XOSA-Average*, *XOSA-One*, and *XOSA-Each*, which differ in terms of what values are passed as arguments to the SA-ACCEPT predicate and in terms of how the current state is updated if acceptance is indicated.

In *XOSA-Average*, the actual parameter for *fitness-of-current-state* is the average fitness of the two current solutions. The actual parameter for *fitness-of-candidate-state* is the average fitness of two candidate solutions derived from twice crossing over the current solutions. If acceptance is indicated, both the candidate solutions replace the current solutions. Since XOSA-Average uses the SA component for each pair of fitness evaluations, the SA component is adjusted to use half as many steps in the cooling schedule.

In *XOSA-One*, only one child is generated, via crossover, from the current solutions. The fitness of the weaker parent is the value for *fitness-of-current-state* and

[6]If more evaluations were allowed the cooling would be slower but would reach the same temperature.

[7]Behind the scenes another component of the SA algorithm changes the system temperature when appropriate.

the fitness of the child is the value for *fitness-of-candidate-state*. The weakest parent is replaced by the child, if acceptance is indicated.

XOSA-Each attempts to address a 2-parent to 2-children function and the option of only accepting one child. If the weakest current solution can be replaced by the fittest child, this is done and then an attempt is made to exchange the fitter current solution for the weakest child. Otherwise, an attempt is made to exchange the weakest parent for the weakest child.

The only parameter of the crossover component of the algorithm is "*xo-tries-limit*". Once this many crossover attempts have been made with the same pair of parents, the weaker parent is replaced by a random program. Like *xo-tries-limit* in $XOHC$ (Section 2.1) this parameter is a sort of patience threshold. We used the same values for it in $XOSA$ as in $XOHC$.

2.3 GP-XOHC and GP-MU-HC Algorithm

The algorithms designed for hybridized GP and hill climbing, *GP + XOHC* and *GP + MU-HC* are simple. Encouragingly, they still perform effectively despite not being adaptive. Every g generations, the f fittest individuals in the population are used as the starting points of a hill climbing search. Each hill climb lasts for e evaluations. The best individual from each hill climb is placed in the next generation and then the remaining individuals of the population for that generation are generated standardly (i.e. with crossover or direct reproduction). For the duration of a run the move operator of the hill climb is either entirely GP Crossover or entirely HVL-Mutate. The parameters g, f, and e are supplied *a priori* to the run. We ran three parameter settings. We always used $f = 5$. When the climb was 500 evaluations, the interval g was either 2 or 5 generations. When the climb was 100 evaluations, the interval g was 3 generations.

The interesting design issue is how to obtain mates for the crossover hill climb. Should one exploit the knowledge embodied by the current population by using its membership as a source of mates? We experimented with 3 options: *Random*, *Best*, and *Population*. With the *Random* option, mates are not drawn from the population at all, but are randomly created. With the *Best* and *Population* options, mates are drawn from the group of individuals with the highest fitness or randomly drawn from the population at large, respectively.

The algorithm must adjust the maximum number of generations to take into account the additional evaluations used by the hill climbing. A maximum which gives a close approximation to the *a priori* given maximum number of fitness evaluations and population size is calculated. In the case of 25500 evaluations and a population of 500 (which is used in every *GP+HC* run), for a run with $f = 5$, $g = 3$ and $e = 100$ the maximum generations is 39 and the maximum evaluations is 26435. For $f = 5$, $g = 2$, $e = 500$, the maximum generations is 15 and the maximum evaluations is 25465. Finally, for $f = 5$, $g = 5$, $e = 500$, the maximum generations is 26 and the maximum evaluations is 25975.

We decided to check for the presence of a perfect individual only at the end of a generation. With this decision it does not matter whether hill climbing is done before or after the normal GP crossover of a generation. One consequence is that the actual number of fitness evaluations reported for successful runs is slightly over-estimated but no more than if a standard GP run were executed and the same check done at the end of each generation.

3 Test Suite and Methodology

We experiment with 5 problems: the 6-bit Boolean Multiplexer (6-Mult), the 11-bit Boolean Multiplexer, (11-Mult), Block Stacking and sorting (Sort-A and Sort-B). The Multiplexer task is to decode address bits and return the data value at the address. 6-Mult uses the primitives IF, OR, NOT, AND which take 3, 2, 1, and 2 arguments respectively. There are 6 variables (i.e. primitives which take no arguments): A0, A1, D0,...,D3 which are bound before execution to the address bits and data values of a test case. All 64 possible configurations of the problem are enumerated as test cases. A program's raw fitness is the number of configurations for which it returns the correct data value for the given address. 11-Mult is simply a larger scale version of 6-Mult using 3 address bits (A0- A2) and 8 data values (D0-D7). The test suite consists of 2048 test cases.

Block Stacking is well explained in [10]. Succinctly, the task is to stack labeled blocks upon a table in correct order according to a given goal list starting from any arbitrary configuration of the stack and remaining blocks on the table. Block Stacking uses 3 "sensors" which are primitives encoded to return state information. All sensor primitives have zero arguments. It also uses 5 primitives which are operators for manipulating blocks. They take either 1 or 2 arguments. A structured sample containing 166 of the possible test cases was used as a test suite. The raw fitness of a program is the number of test cases for which the stack is correct.

The task of a sorting program is to arrange the elements of an array in ascending order. A description of the primitives used is in [11]. A program is run 48 times, each time with a different array bound to the primitive *array*. The arrays in the test suite range in size and sorted order. In Sort-A the raw fitness of a program is the sum of the number of elements found in the correct position after running the program. In Sort-B the raw fitness is the summed permutation order [9] of each array after each execution. The intention of experimenting with two different fitness functions and the same repertoire of primitives is to isolate the impact of specific fitness functions on a fitness landscape.

In order to compare results among GP, *GP+HC*, *XOHC* and *XOSA*, each run was permitted the same maximum number of evaluations. Our benchmark GP runs were run with a population of 500 for 50 generations which (given fitness evaluations for the initial generation) sums to a maximum of 25500 fitness evaluations per run. This approximate number of evaluations was used for *GP+HC* runs (see Section 2.3 for details) and *XOSA* and *XOHC* were given a precise maximum of 25500 evaluations. At least 30 runs of each problem were executed. A run is deemed successful if an individual scores the maximum fitness. The fitness values in the GP and *GP+HC* runs for all five problems were scaled by linear and exponential factors of 2. In standard GP reproduction (i.e., not during hill climbing) the crossover operator was applied 90% of the time with the remaining 10% of individuals chosen by selection being directly copied into the next generation. We used a T-test with 95% confidence to determine if runs were significantly different.

Regarding tabular data, the column "Evals Used" expresses how many evaluation were used as a percentage of the actual maximum evaluations allowed each run. The column "Fittest Individual" is an average of the best fitness of each run taken as a percentage of the perfect fitness. Standard deviation figures are in parentheses. Where relevant, previous results of HVL-Mutate Hill Climbing, "*MU-HC*", HVL-Mutate Simulated Annealing, "$MU-SA$", and GP experiments with other crossover operators are included. The standard GP Crossover is abbreviated at "GP-XO". "Ht-Fair-XO" groups subtrees by height, selects one group and one subtree from that group at random for exchange. "Fair-XO" selects among subtrees with equal probability.

4 Experimental Results

Table 1: 6-Bit Boolean Multiplexer

6 Bit Boolean Multiplexer	Prob of Success (%)	Fittest Individual (%)	Evals Used (%)	Evals Used in Succ Run (%)
GP with Ht-Fair-XO	86.7 (34.0)	98.3 (9.5)	55.2	51.4
GP with GP-XO	79.5 (40.3)	98.0 (4.5)	55.7	48.4
$XOHC$, xo-tries-limit = 100	100.0	100.0	20.56	20.56 (14.2)
$XOHC$, xo-tries-limit = 1000	96.7 (17.8)	99.5	23.2	23.2
$XOSA-Each$	36.7 (48.2)	91.6	92.0	80.5
$XOSA-One$	16.7 (37.3)	64.6	94.8	70.0
$XOSA-Ave$	3.3 (17.9)	65.7	99.2	87.0
$MU-HC$, xo-tries-limit = 500	40.0 (49.0)	96.7	81.8	61.3
$MU-HC$, xo-tries-limit = 10K	77.0 (50.4)	98.8	61.5	57.7
$MU-SA$	100.0	100.0	54.0	54.0 (7.2)
$GP+MU-HC$, $f=5$				
$e=500, g=5$	93.3 (25.0)	99.5	45.2	43.3 (22.5)
$e=500, g=2$	90.0 (30.0)	99.5	32.5	30.6 (5.1)
$e=100, g=3$	80.0 (40.0)	99.8	57.0	47.7 (21.3)
$GP+XOHC$ $f=5$, $e=100$, $g=3$				
$Best$	50.0 (50.0)	96.8 (3.7)	74.0	50.0 (24.9)
$Random$	60.0 (14.7)	96.9 (4.2)	72.1 (29.6)	32.4 (17.9)
Pop	63.3 (14.5)	95.7 (4.2)	65.3 (32.5)	44.9 (20.8)
$GP+XOHC$ $f=5$, $e=500$, $g=2$				
$Best$	86.7 (10.2)	99.4 (1.7)	44.7	32.4 (17.9)
$Random$	93.3 (7.5)	99.7 (1.2)	45.9	44.9 (17.7)
Pop	100.0	100.0	31.3	31.3 (17.4)

6 Bit Boolean Multiplexer: Table 1 summarizes the results for 6-Mult. This is clearly a relatively easy problem. Three of the algorithms stood out significantly by solving the problem 100% of the time. These were $XOHC$, *MU-SA* and *GP+XOHC-population.* Among these 3, the ranking in terms of evaluations used to find the perfect solution was: 1)$XOHC$, 2)$GP + XOHC$, 3)$MU - SA$.

The improvement in the probability of success due to the hill climbing component being added to GP was statistically significant (15 - 20%). In the GP experiments, the runs with higher probability of success unfortunately required more evaluations than the runs with lower probability of success. The fitness evaluations of *GP+XOHC* were significantly 10 - 20% lower than GP. Thus, *GP+HC* was not only more reliable but also less computationally expensive.

Table 2: 11-Bit Boolean Multiplexer

11 Bit Boolean Multiplexer	Prob of Success (%)	Best of All Runs (%)	Fittest Ind (%)	Evals Used (%)	Evals Used in Succ Run (%)
GP with Ht-Fair-XO	0	93.8	80.9	100	
GP with GP-XO	0	87.6	79.2	100	
$XOHC$, xo-tries-limit=100	16.7	100.0	93.6	95.4	72.4
$XOHC$, xo-tries-limit=5000	16.7	100.0	94.4	93.3	60.0
$XOSA - Each$	0	75.0	66.5	100.0	
$XOSA - One$	0	78.9	68.5	100.0	
$XOSA - Ave$	0	81.3	71.7	100.0	
$MU - HC$					
xo-tries-limit = 500	0	84.4	81.6	100.0	
xo-tries-limit = 5000	0	81.3	71.7	100.0	
$MU - SA$	16.7	100.0	93.0	99.5	98.2
$GP + MU - XO, f = 5$					
$e = 500, g = 5$	0	91.9	89.6	100.0	
$e = 500, g = 2$	3.3	100.0	91.0	99.5	48.6
$GP + XOHC, f = 5, e = 500, g = 2$					
Best	0	91.9	89.6	100.0	
Random	3.3	100.0	90.2	99.8	97.9
Population	0	91.4	85.0	100.0	

11 Bit Boolean Multiplexer: Once Hill Climbing was combined with GP, it was finally possible to find a perfect solution to the 11-Mult in 25500 fitness evaluations. This had not ever been done with GP alone. While, the GP hybrids did yield an improvement over GP, they did not better $XOHC$ and *MU-SA* which were (significantly) the best. Of the new crossover algorithms, $XOHC$ was equal in terms of probability of success to *MU-SA*. $XOSA$ performed the worst of all algorithms.

Sorting: The Sorting search landscapes appear to differ in some salient characteristic(s) from the group of 6-Mult, 11-Mult and Block Stacking. We conjecture this because the hybrid *GP-XOHC* algorithm was not an improvement for Sort-A or Sort-B and both problems were the only ones upon which $XOSA$ performed encouragingly. One *GP+MU-HC* hybrid (e = 100, g = 3) solved both Sort-A and Sort-B 100% of the time. This was a significant improvement over GP alone and the best algorithm for the sorting problems.

Block Stacking: Phrased in its present manner, Block Stacking is a very easy problem for every algorithm we tried, except GP! Among the algorithms that solved all runs, based upon the number of fitness evaluations used, the ranking (with statistical significance) was 1) $XOHC$, 2) *GP+XOHC*, 3) *GP+MU-HC*, and 4) $MU - SA$.

All GP hybrids reached 100% success rates. On *GP+MU-HC* ($e = 500$, $g = 2$) 83.3% of the runs solved the problem via hill climbing. In fact, on quite a few runs, more than one hill climb (av. 3.4) found the perfect solution. For $e = 100$, $g = 3$ there were, on average 1.8 successful hill climbs over the 73.3% of the runs that converged via hill climbing. The *GP-XOHC* runs had similar statistics.

Table 3: Sort-A and Sort-B

	Sort-A			Sort-B		
Sort-A and Sort-B	Prob of Success (%)	Evals Used (%)	Evals Used in Succ Run (%)	Prob of Success (%)	Evals Used (%)	Evals Used in Succ Run (%)
GP with Fair-XO	73.3 (45.0)	62.1	48.3 (31.4)	93.3 (25.4)	38.2 (29.3)	33.8 (24.3)
GP with GP-XO	80.0 (40.0)	51.3	39.1 (26.6)	67.7 (47.1)	60.4 (37.5)	40.6 (28.2)
XOHC, xo-tries-limit= 50	10.0 (30.0)	92.3	22.7 (22.5)	40.0	81.4	53.6 (31.1)
$XOSA - Each$	70.0 (45.8)	56.8	55.3 (25.5)	80.0	58.9	48.75 (25.1)
$XOSA - One$	93.3 (25.0)	25.1	22.5 (22.5)	83.3	48.4	38.3 (30.9)
$XOSA - Ave$	66.7 (47.1)	67.8	52.0 (31.5)	90.0	65.3	61.5 (17.9)
$MU - SA$	83.3 (37.9)	66.2	54.6 (28.5)	88.3	64.0 (30.5)	56.6 (28.5)
$GP + MU - HC$, $f = 5$						
$e = 500, g = 2$	93.3 (25.0)	32.5	11.3 (20.0)	93.3	32.3	30.9 (30.0)
$e = 100, g = 3$	100.0	42.0	42.0 (26.9)	100.0	40.3	40.3 (27.2)
$GP + XOHC$, Pop, $f = 5$						
$e = 500, g = 2$	70.0 (45.8)	58.4	43.4 (18.2)	66.7	58.8 (31.0)	44.3 (18.4)
$e = 100, g = 3$	80.0 (40.0)	50.6	39.6 (26.6)	80.0	50.8 (34.0)	40.0 (26.2))

Table 4: Block Stacking

Block Stacking	Prob of Success (%)	Fittest Individual (%)	Evals Used (%)	Evals Used in Succ Run (%)
GP with GP-XO	76.7 (16.3)	87.7	43.7	35.1 (26.6)
XOHC, xo-tries-limit=250	100.0	100.0	2.6	2.6 (2.3)
$XOSA - Each$	70.0 (45.8)	87.7	75.1	64.7 (26.5)
$XOSA - One$	96.7 (17.8)	98.4	28.3	25.8 (27.1)
$XOSA - Ave$	60.0 (49.0)	84.5	88.4	80.8 (18.96)
$MU - HC$, max-mu=100	94.3 (23.2)	94.9	34.5	30.5 (24.0)
$MU - SA$	100.0	100.0	28.5 (22.2)	28.5
$GP + MU - HC$, $f = 5$				
$e = 500, g = 2$	100.0	100.0	5.3	5.3 (3.3)
$e = 100, g = 3$	100.0	100.0	12.5	12.5 (10.9)
$GP + XOHC$, Pop, $f = 5$				
$e = 500, g = 2$	100.0	100.0	5.3	5.3 (3.3)
$e = 100, g = 3$	100.0	100.0	13.1	13.1 (11.8)
$GP + XOHC$, Random, $f = 5$				
$e = 100, g = 3$	100.0	100.0	10.6	10.6 (6.3)

4.1 Summary of Algorithms

XOHC is conclusively a search algorithm worth consideration. It outperformed or equaled the best of the other algorithms on 6-Mult, 11-Mult and Block Stacking.

However, its poor performance on Sorting is a reminder that no search algorithm can be expected to always be superior.

Given the strong performance of *XOHC*, *XOSA* was predicted to have merit but its results did not. Perhaps, an adjustment in the temperature schedule or *xo-tries-limit* value might be in order. Given the robust performance of GP+HC hybrids and *XOHC*, it seems an unlikely algorithm to use again. On each problem, except Sort-B, there was significant difference among the three versions (*Average, Each, One*). While *Average* was never solely best, *Best* and *Each* exchanged rankings on different problems.

Among the Hill Climbing and SA algorithms, we are still somewhat confused that, on any given problem, neither both algorithms with the same operator nor both operators with the same algorithm had correlated performance. Our preliminary conjecture is that since HVL-Mutate has a larger search neighbourhood than crossover, SA works better than hill climbing because it does not have to explore the neighbourhood thoroughly (i.e. it accepts bad moves). Since XO has a smaller search neighbourhood, hill climbing is better than SA because time is not lost doing many wasted evaluations, on a landscape with few local optima. In such a landscape, hill climbing is perfectly exploitative whereas SA is too explorative.

The hybrid of GP plus Hill Climbing was better than GP alone **on all problems, in both forms -** *GP+XOHC*, *GP+MU-HC*, with at least one parameter setting. Prior to using this search strategy, GP had not been superior nor sometimes even on par with SA and Hill Climbing. With hybridization the evolution inspired search model is, at the least, comparable. We observed no consistent significant ordering between the mutation hill climbing option or crossover hill climbing option across the testsuite.

Regarding the relative merits of the *Random*, *Best*, and *Population* options of *GP+XOHC*, current data is not decisive. Qualitatively, *Best* may not be explorative enough because it is limited to a mate pool that may be very small. Preliminarily, *Best* was outperformed on 6-Mult but comparable on 11-Mult. *Population* worked better than *Random* on 6-Mult but the results were reversed on 11-Mult. On Block Stacking and both sorting problems *Random* and *Population* were equal.

5 Conclusion and Future Work

Comparison is a vital part of evolution-based program discovery search research. By mixing and matching operators and search strategies we have produced new algorithms that improve upon existing ones. We have started to differentiate among our small suite of problems based upon the response of different algorithms to them. That encourages us to seek quantitative measures of correlation between search landscapes and search algorithms. In particular, we are pursuing characterizing the genotypic distance distribution in search operator neighbourhoods.

Acknowledgments

U.M.O'R. wishes to especially thank Terry Jones for many useful discussions. She also thanks the members of the Adaptive Computation program at the Santa Fe Institute, researchers at SFI, and members of Stephanie Forrest's graduate student group.

References

[1] Aarts, E., Korst, J., Simulated Annealing and Boltzmann Machines. Wiley. 1989

[2] Chen, H., Flann, N., *Parallel Simulated Annealing and Genetic Algorithms: A Space of Hybrid Methods*, Parallel Problem Solving from Nature III, Davidor, Schwefel, Manner (Eds), Springer Verlag (LNCS), Berlin,1994.

[3] Davis, L. (Ed), Genetic Algorithms and Simulated Annealing, Morgan Kaufmann, CA, 1987.

[4] de Souza, P., Talukdar, S., *Genetic Algorithms in Asynchronous Teams*, Proceedings of 4th International Conference on Genetic Algorithms, R. Belew, L. Booker (Eds), Morgan Kaufmann, CA, 1991.

[5] Jones, Terry. *Crossover Hillclimbing and the Dual Role of Crossover in Genetic Algorithms*, submitted to ICGA-95.

[6] Jones, Terry. PhD. Dissertation in preparation, forthcoming in 1995.

[7] Jones, Terry. Personal Communication.

[8] Kido, T., Kitano, H., Nakashani, M., *A Hybrid Search for Genetic Algorithms: Combining Genetic Algorithms, TABU Search, and Simulated Annealing*, Proceedings of 5th International Conference on Genetic Algorithms, S. Forrest (Ed), Morgan Kaufmann, CA, 1993.

[9] Knuth, D.E., The Art of Computer Programming, Addison-Wesley, MA, 1968.

[10] Koza, J. R., Genetic Programming; On the Programming of Computers by Means of Natural Selection. Bradford Books, 1992.

[11] O'Reilly, U. M. and F. Oppacher (1994). *Program Search with a Hierarchical Variable Length Representation: Genetic Programming, Simulated Annealing and Hill Climbing*, Parallel Problem Solving from Nature III, Davidor, Schwefel, Manner (Eds), Springer Verlag (LNCS), Berlin,1994.

[12] Press, W. H.: Numerical Recipes in C: the art of scientific computing. Cambridge University Press, 1992.