CHAPTER 3

Crossover, Macromutation, and Population-based Search

3.1. Introduction

The features of a GA that distinguish it most from other search methods are its use of a population and a crossover operator. In a GA, crossover is the major mechanism of communication between individuals in a population. The usefulness of crossover, and therefore, to some extent, of maintaining a population, has been a contentious issue in research on GAs (see, for example, [29, 88, 89, 90, 91, 92]). The standard method of determining whether crossover is useful to a GA involves comparing a GA with crossover to a GA without crossover. If the GA with crossover outperforms the GA without crossover, this is taken as evidence that crossover is useful (i.e., that it is facilitating the exchange of building blocks between individuals). Therefore, the maintenance of a population and the use of crossover seem justified because these are clearly an aid to the algorithm. This chapter argues that this conclusion is not justified and gives an alternate method for assessing the utility of crossover.

A distinction is made between the *idea* of crossover and the *mechanics* by which crossover operators attempt to implement this idea. It is shown that even in the absence of the idea, the mechanics alone can be effectively used for search. A GA with no crossover has neither the idea nor the mechanics of crossover, and if it is outperformed by a normal GA, it does not follow that the idea of crossover is therefore useful to the normal GA. An algorithm is presented that illustrates the power of the mechanics of crossover. This suggests a simple new comparison for determining whether crossover and a population are providing some additional benefit to the GA over simpler algorithms that do not use crossover or maintain a population. This comparison is between a normal GA and a GA that uses *random crossover*. Using this, it possible to see what advantage the GA is gaining from the idea and the mechanics of crossover over and above what it could be gaining from the mechanics alone. In some cases, when well-defined building blocks are not present, the GA may actually perform *worse* with normal crossover than a GA with random crossover because of its use of a population. This test gives an indication of the existence of building blocks that are exploitable by a GA with a given crossover operator.

3.2. Search as Navigation and Structure

The model of landscapes presented in Chapter 2 maintains that every operator used in a GA creates its own landscape, and advocates a view of search as a process of navigation upon these landscapes. This division into structure and navigation invites the construction of new algorithms. A potentially new search algorithm may be constructed from two existing search algorithms by combining the navigation strategy of one algorithm with the landscape structure(s) from the other. This observation prompted consideration of combinations involving the representation used in genetic programming and navigation strategies of hillclimbing and simulated annealing by O'Reilly and Oppacher [93].

This process is possible because navigation strategies are designed to search graphs and landscapes are graphs. This implies that the pieces will be interchangeable provided the graph satisfies any assumptions made by the navigation strategy (e.g., that it is a tree). The resulting search algorithm may be interesting in its own right (for instance because it performs better than one or both of the originals in some circumstances) or it may be interesting because it throws light on one or both of the original algorithms. In this chapter the combination of the navigation strategy of a simple hillclimber with the landscape structure of crossover is examined in detail. The resulting *crossover hillclimbing* (CH) algorithm is interesting for both of these reasons—it outperforms both the GA and the hillclimber, and, an investigation into the power of the algorithm leads to a clearer understanding of the dual role of the crossover operator in a GA. This combination of navigation strategy and operator has also been investigated, in the context of genetic programming, by O'Reilly and Oppacher [94].

Navigation strategy has deliberately not been well defined. This allows us to apply the spirit of this division (into navigation strategy and structure) to a very wide range of algorithms, without requiring a specific definition of what a navigation strategy actually is. Landscape graphs on which the algorithms make moves are well defined in §2.4(25). We will define navigation strategy as any part of a search algorithm that is not a landscape. For example, consider a simple hillclimbing algorithm that operates on binary strings using the bit-flipping operator, β, described in §2.6.1(31). Each of the vertices in the landscape graph of β corresponds to a binary string, and each has an associated fitness. If we suppose that the algorithm starts operation at a randomly chosen vertex, v, to which vertex of $N_\beta(v)$ does it move next? This decision is made by the navigation strategy. Notice that we have assumed the algorithm has made the choice to use the operator to generate a new binary string. This choice is part of the navigation strategy, though in such a simple algorithm there is not a lot of choice about what to do next if the navigation strategy does not halt the search. The choice of the next vertex to move to is made by the navigation strategy, so is the number of elements of $N_\beta(v)$ to examine before making the move. Having examined a number of next possibilities, the navigation strategy selects one, usually on the basis of the fitnesses observed.

The particular solutions to the choices above are usually responsible for the name given to the algorithm. For example, if all the neighbors are examined and one of those with maximal fitness is chosen to move to, the hillclimbing algorithm is called *Steepest Ascent*. If the navigation strategy examines neighbors until one with better fitness is found and then moves to it, we shall call the hillclimbing algorithm *Any Ascent*. Other aspects of algorithms that will be considered part of the navigation strategy include such things as: Deciding when to stop, deciding which operators to use and when, deciding how and when to change the algorithm's temperature variable (should one exist), deciding to adjust the representation or the mapping Γ between the object space ($\mathcal{O}$) and representation space ($\mathcal{R}$) to focus on a specific area of $\mathcal{O}$, deciding on population size (if the algorithm maintains a population), and deciding on a balance between exploration and exploitation.

3.3. Crossover Landscapes

In order to understand the workings of the crossover hillclimbing algorithm, it is necessary to have an idea of the structure of crossover landscapes. As was briefly mentioned in Chapter 2, crossover landscapes are not simple connected graphs. This dissertation does not attempt a full study of the structural properties of crossover landscapes, however, several crossover landscapes are shown in Figures 8 to 11. In each of these, $\mathcal{R}$ is the space of binary strings of length two or three. Figure 8 is identical to Figure 4 on page 35 and is reproduced here so as to have these figures close to each other for easy comparison. It shows the landscape generated by the one-point crossover operator that produces two children from two parents ($\chi_1^{2\to2}$). Figure 9 shows the landscape generated by two-point crossover that produces two children from two parents ($\mathcal{X}_2^{2\to 2}$), Figure 10 shows the landscape generated by uniform crossover that produces two children from two parents ($\mathcal{X}_u^{2\to 2}$), and Figure 11 is the landscape generated by one-point crossover that produces one child from two parents ($\mathcal{X}_1^{2\to 1}$). A more general figure for parameterized uniform crossover [95] would have probabilities on the edges, similar to those in Figure 3 on page 33.

3.4. Experiment Overview

The crossover hillclimbing algorithm was constructed to study the performance of crossover in relative isolation. One problem with assessing the importance of the operator is that it is difficult to dissociate the operator and its effects from the rest of the machinery of the GA. Naturally, the power of the operator, if any, might arise from its actions in concert with other operators and this needs to be kept in mind. However, the crossover hillclimber allows a reasonably simple view of crossover and the results obtained with the algorithm throw light on the GA itself.

The CH algorithm is compared on a number of functions to three algorithms: Two simple GAs (one with and one without elitism) and a simple mutational hillclimber. This collection of algorithms is interesting as the two hillclimbing algorithms use a single (different) operator and these two operators are both employed by the GAs. Otherwise, the two hillclimbers have very similar navigation strategies. The GAs of course have a much more sophisticated navigation strategy. The algorithms are compared exclusively on the basis of the expected number of evaluations they require to reach certain levels of performance.[1] It should be emphasized that the purpose of this experiment was *not* to do an exhaustive comparison of the four algorithms. This would be an extremely large undertaking as there are many variants of the GAs and many parameters that could be adjusted. Instead, a fairly standard collection of parameter settings for these algorithms was chosen and held fixed across all experiments.

On several occasions I performed some initial testing to see whether a particular parameter setting appeared to increase an algorithm's overall performance. In each case I chose the option that resulted in the higher performance. Virtually all this parameter testing was for the GAs, though it was necessary to change the mutational hillclimbing *algorithm* (from the description given by Forrest et al. [96]) to increase its

[1] This is a practical yardstick, and is motivated by the belief that when choosing an algorithm to solve a problem, one is chiefly motivated by the amount of time that will be spent awaiting a solution. If the number of function evaluations performed by an algorithm is proportional to its run time, then concentrating on the expected number of evaluations is appropriate.

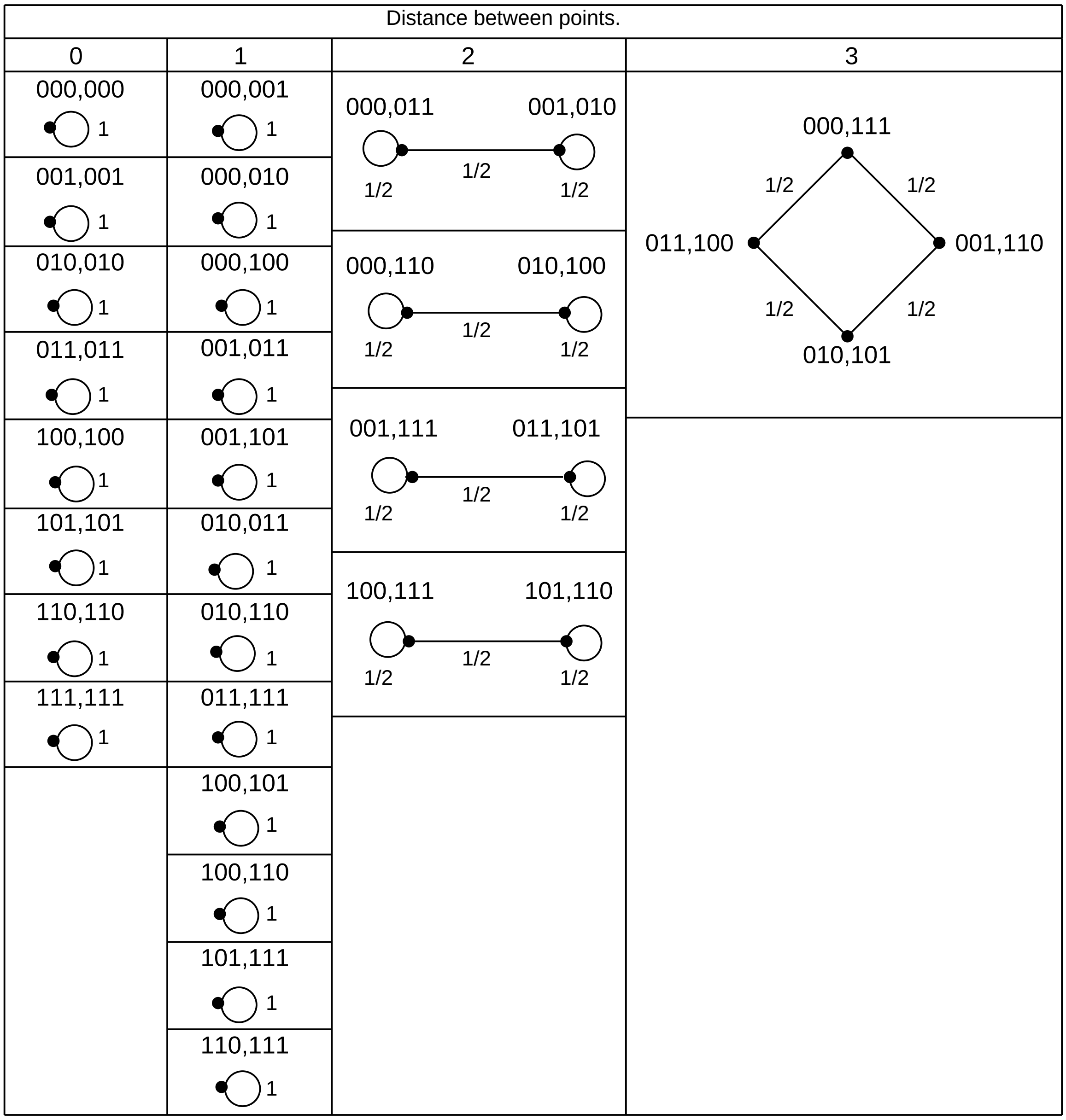

Figure 8. The one-point crossover landscape for binary strings of length three. The crossover operator, $\chi_1^{2\rightarrow 2}$, produces two offspring from two parents. Edges are bidirectional.

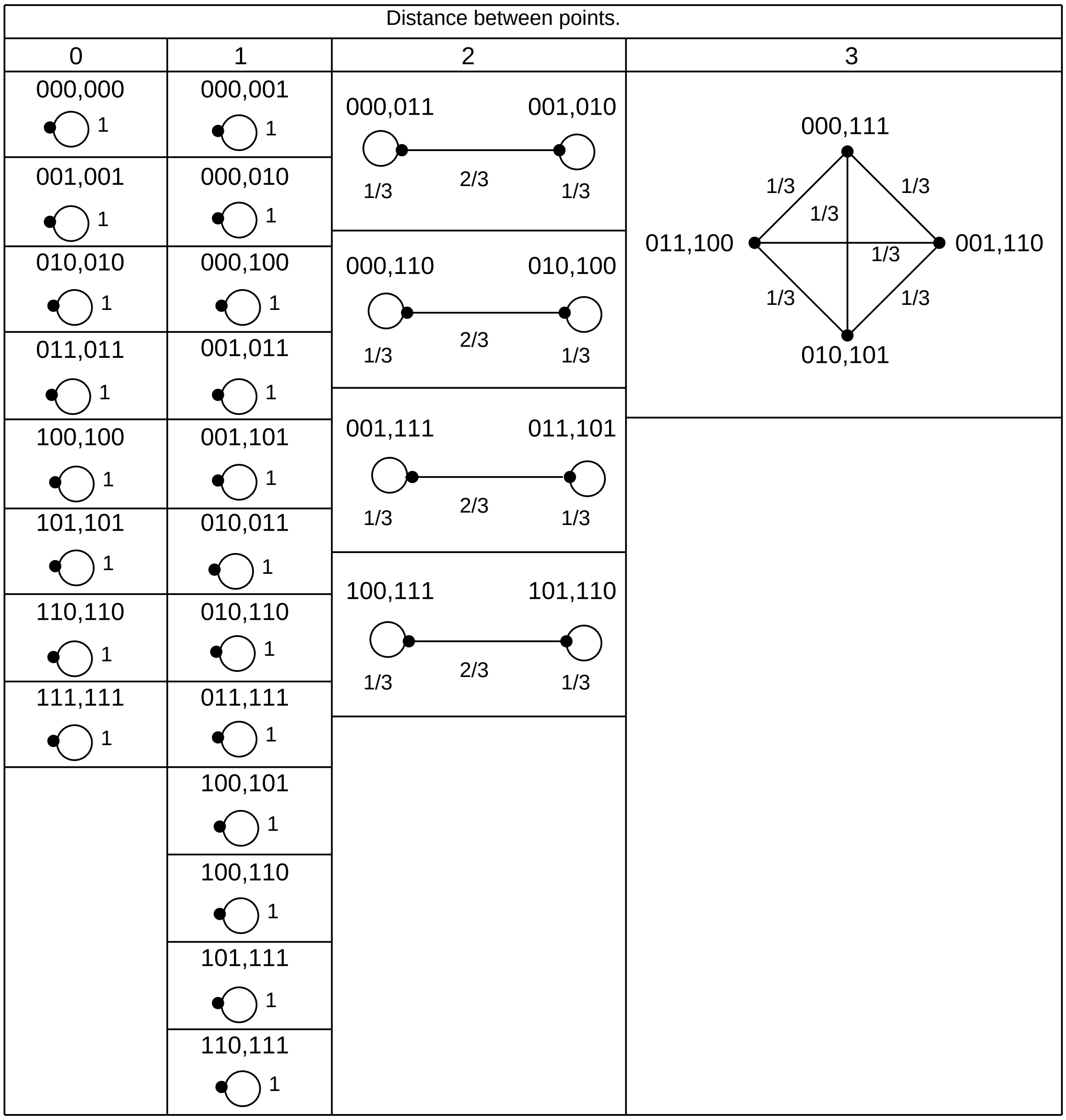

Figure 9. The two-point crossover landscape for binary strings of length three. The crossover operator, $\chi_2^{2\to2}$, produces two offspring from two parents. Edges are bidirectional.

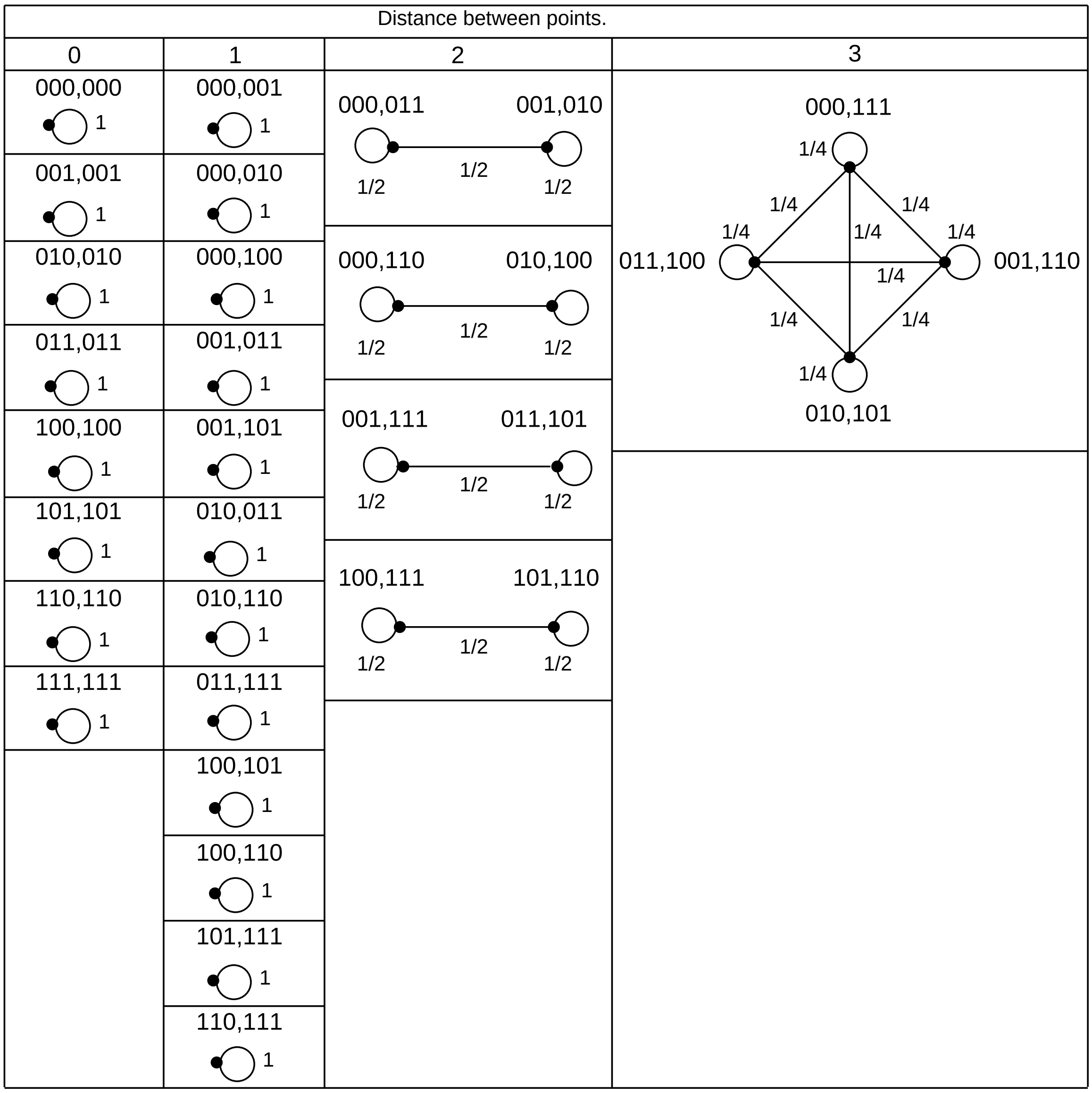

Figure 10. The uniform crossover landscape for binary strings of length three. Bits are chosen from parents with probability 1/2. The crossover operator, $\chi_u^{2\to 2}$, produces two offspring from two parents. Edges are bidirectional.

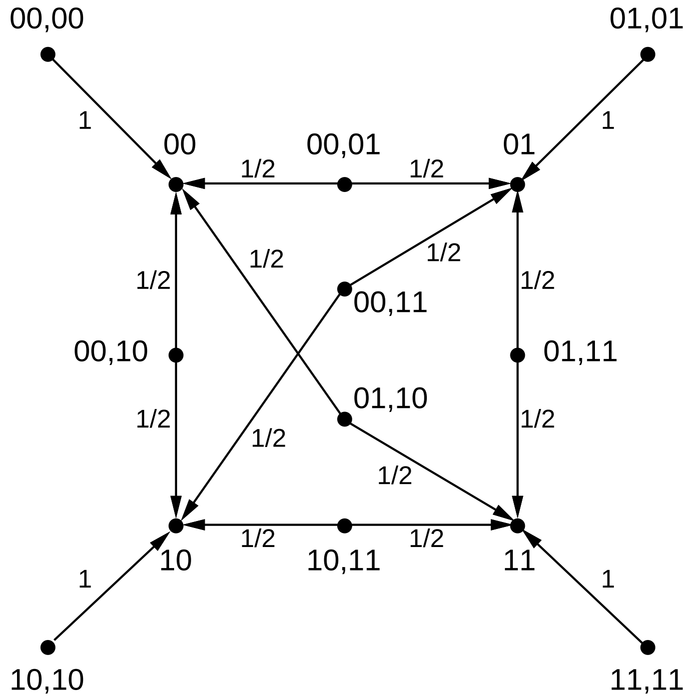

Figure 11. The one-point crossover landscape for binary strings of length two. The crossover operator, $\chi_1^{2\rightarrow 1}$, produces one offspring from two parents (and is consequently not walkable). This is done by generating both offspring (as in $\chi_1^{2\rightarrow 2}$) and then selecting one uniformly at random to retain. Other offspring selection methods would alter edge probabilities.

performance. This is described in §3.6(55). The crossover hillclimber was surprising in its robustness. The parameter settings used were chosen after a few very preliminary experiments and not altered until the reasons for the success of the algorithm were later sought.

3.5. The Simple Genetic Algorithm

The following parameter settings were used in the GAs. These can all be defended as fairly standard. Population size was 500 (initial experiments with smaller sizes resulted in a marked decrease in the GA's performance). The maximum number of generations was 100. This many generations were run unless the population became too converged or a point with maximal fitness was encountered. The maximum amount of convergence permitted in a population was 0·85 at every locus. If every locus in the population converges to at least this frequency, the run is terminated.

The per-locus mutation probability was 0·01. The probability that crossover was applied to a pair before they were added to the next generation was 0·75. To avoid complications with selection, binary tournament selection [97] was used as it is rank-based, avoiding any need to scale population fitness or otherwise consider the fitness function, even across experiments. The winner of the tournament was given a 0·75 chance of being selected. See [98, 99] for discussion on the advantages of rank-based selection schemes.

A major omission from this list of parameters is some form of elitism [30], either an explicit copying of the best individual or a generation gap less than one to decrease the probability of discarding the best structure found so far. There has been recent discussion on the use of GAs as function optimizers [100, 101], perhaps stemming from Holland's reminder that the original intent of the GA was more one of improvement than one of optimization [102]. It is not clear whether a standard GA should include some form of elitism or not [5, 9, 30, 103]. Grefenstette's standard GA [103], which takes its lead from De Jong [30] recommends some form of elitism. De Jong refers to a "canonical" GA (which does not use elitism) and mentions elitism as a method of improving GA performance when a GA is used as a function optimizer [100].

It is perhaps unfair to compare the GA against two algorithms that use a very elitist hillclimbing strategy, but then again, they *are* different algorithms. The obvious solution is to examine both possibilities. A summary of the result of this is that elitism never resulted in worse performance (usually the GA with elitism would attain difficult performance levels in five to ten times fewer function evaluations than the GA without elitism), but that the change never resulted in a significant change in the positional ranking of the GA with respect to the other two algorithms. For example, if the performance on a problem of a GA without elitism was between the other algorithms, then the GA with elitism would perform better than the GA without elitism, but still between the other algorithms. In what follows, the standard and elitist versions of the GA are denoted by GA-S and GA-E. Abbreviations for all the algorithms of this chapter can be found in Table 1.

3.6. The Bit-flipping Hillclimbing Algorithm

The bit-flipping hillclimbing algorithm (BH) is essentially identical to the "random mutation hillclimbing" (RMHC) algorithm suggested to Forrest et al. [96] by Richard Palmer. The algorithm chooses a random starting vertex and thereafter proceeds to any β-neighbor with equal or higher fitness as soon as one can be located. On the royal road functions examined by Mitchell, Forrest and Holland [36, 96], the algorithm

Table 1. Algorithm abbreviations and brief descriptions. The last column gives the page on which a fuller description of each algorithm can be found.

Abbreviation	Description	Page
GA-S	Standard GA.	54
GA-E	Elitist GA.	54
BH	Random Mutation Hillclimbing.	55
CH(a,s,j)	Crossover Hillclimbing, a attempts, s steps, j jumps.	57
CH	CH(10,3,1000).	57
CH-1S	CH(10,1,3000) One step per hypercube.	73
CH-NJ	CH(10,3000,0) No jumps between hypercubes.	74
GA-RC	GA with random crossover.	77
GA-RCE	Elitist GA with random crossover.	77
BH-MM	Hillclimbing with macromutations.	81
BH-DMM	Hillclimbing with distributed macromutations.	81

has the important property of moving indefinitely on a β-plateau. The royal road functions have many such sets, which are not β-mesas and so are eventually escaped by this algorithm. This accounts for its success on these problems over other hillclimbers that use β but do not make moves to points of equal fitness.

In general, the operators and representation chosen to attack a search problem may create many plateaus that *are* mesas. When BH encounters a β-mesa it will wander on it indefinitely. For this reason, the algorithm has a parameter that limits the number of steps that may be taken without a fitness increase. This parameter is the only difference between BH and RMHC. If this number (which was set to 10,000) is exceeded, the search is terminated. Without such a limit, BH encounters chronic problems on the busy beaver problem (described below) as a result of the choice of representation for that problem. The maximum number of random mutations to try before deciding that a point has no equal or higher neighbor and terminating the search was set to twice the number of neighbors.

3.7. The Crossover Hillclimbing Algorithm

The CH algorithm has a navigation strategy that is very similar to that of the BH algorithm just described. However, the operator used here is two-point crossover (in which two offspring are produced from two children).

The two-point crossover landscape on binary strings described in §3.3(49) is a family of hypercubes, each of whose vertices corresponds to a pair of individuals from $\mathcal{R}$. To conduct a hillclimb within a hypercube, randomly choose a starting pair (A_0, B_0) and perform crossover on them repeatedly until some limit (*max-attempts*) is reached or a pair (A_1, B_1) is found with $\max(f(A_1), f(B_1)) \geq \max(f(A_0), f(B_0))$. If no such pair is found, halt the hillclimb. Otherwise, continue, performing a similar search from (A_1, B_1). This process carries out a single climb within a hypercube of the landscape.[2] Once *max-attempts* crossovers have been tried and a new vertex containing a better individual has not been found, there is no need to discard both A_i and B_i to continue the search. Instead, the CH algorithm discards the less fit of A_i and B_i (ties are broken arbitrarily) and replaces it with a randomly chosen element of $\mathcal{R}$. This has the effect of (probably) transferring the search into another hypercube, from which the hillclimb may continue in identical fashion. Figure 12 shows a fragment of such a climb. A second parameter, *max-jumps* limits the number of times the jumping between hypercubes may occur. A third, *max-steps* can be used to limit the number of steps that may be taken consecutively within a hypercube before a jump is forced. This is equivalent to saying that the number of pairs in the sequence of steps $(A_0, B_0), (A_1, B_1), \cdots$ within a hypercube is limited to $max - steps + 1$. A triple representing the value of these three parameters, *max-attempts*, *max-steps* and *max-jumps* can be used to describe a particular CH algorithm. For example, CH(10,3,1000) represents the CH algorithm that examines up to 10 neighbors (under crossover), limits the number of steps that may be consecutively taken within a hypercube to three, and makes at most 1000 jumps via discarding the worst individual of the current pair when no acceptable crossover is found or when three steps have been taken within a hypercube. In the initial experiments of this chapter, we will be concerned only with CH(10,3,1000) and will use CH to mean this algorithm.

---

[2] Naturally, there is a chance (typically high) that the global optimum (assuming there is only one) cannot be discovered from a randomly chosen starting pair. With binary strings of length l, this probability is $1 - (3/4)^l$.

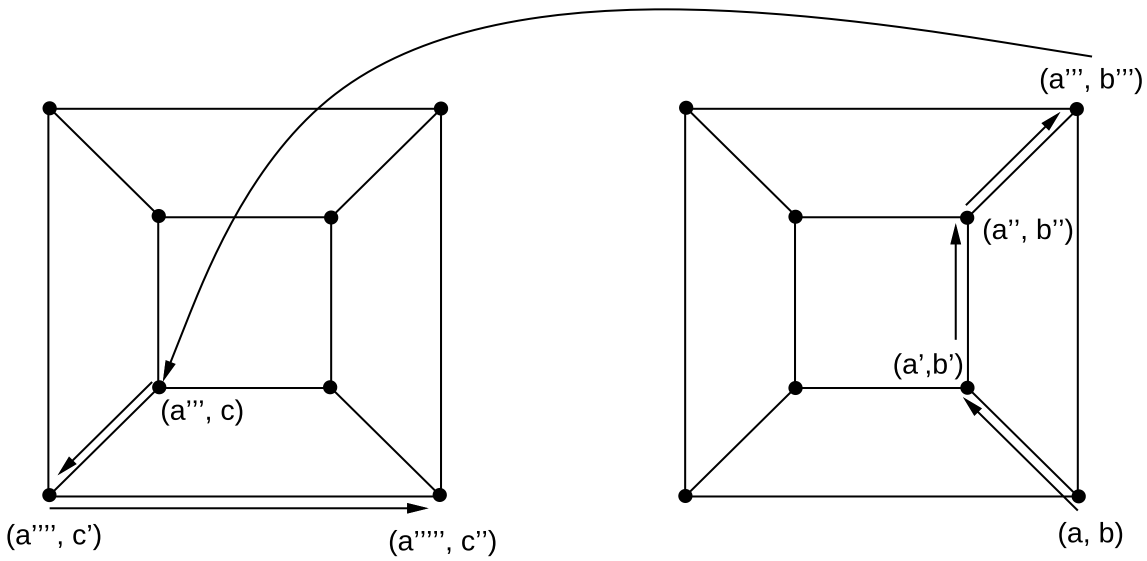

Figure 12. A fragment of a hillclimb under crossover. Three uphill steps are taken (using crossover) within the right hypercube, from (a, b) to (a''', b'''). At that point b''' is discarded and replaced with the randomly generated c. Two more steps are then taken on the left hypercube. The jump into the left hypercube is either the result of a limit on the number of steps in a hypercube (max-steps) or a limit on the number of attempts to find an improving crossover (max-attempts).

3.8. Test Problems

This section describes the set of six problems that were studied. Each of these problems has a fixed length string representation. Three instances of each of the six problems were studied, resulting in eighteen separate problems in six classes.

3.8.1. One Max

This problem was studied using a GA (and other algorithms) by Ackley in [26]. The task is simply to locate a binary string which consists of as many ones as possible. The fitness of a string is the number of ones it contains (Ackley used ten times this). Problems in which the fitness of an individual is a function of the number of ones in the string have been termed *functions of unitation* by Goldberg and have received a considerable amount of attention, particularly in the study of deception [104]. While the problem clearly has a single global maximum and no β-local-optima, there are local optima for other operators, as demonstrated by Culberson [29], who shows that the problem contains $\chi_1^{2\rightarrow 2}$-local-optima. The three instances of the problem that were studied had lengths 30, 60 and 120.

3.8.2. Fully Easy

This problem requires the solution of a number of fully easy subproblems. Each subproblem is a six bit function of unitation. The function values for the different unitation values are arranged in such a way that all low-order schema effectively point the way to the global optimum. This function was designed by Deb and Goldberg [105]. The function values according to unitation (number of ones present in the string) are shown in Table 2.

Table 2. A fully easy 6-bit problem. Unitation is the number of bits in the binary string that are set to one. The maximum fitness point is the string 000000 with zero ones.

Unitation	0	1	2	3	4	5	6
Fitness	1·0	0·8	0·6	0·9	0·5	0·7	0·9

Fully easy problems were designed with GAs in mind, and their name reflects reasoning that they should be easy for GAs to solve. This does not imply that they will be easy for other forms of search—by observation, the problem contains β-local-maxima. For instance, a steepest ascent hillclimber using β would reach a local maximum if it started to climb from any string with more than a single one bit. The global optimum, the string with six zeroes, has a basin of attraction that includes only seven of the sixty-four possible strings. Each six bit problem has $\binom{6}{3} + \binom{6}{6} = 21$ β-local-maxima, which is almost a third of the entire space. A discussion of the problem's difficulty for BH can be found below. Each problem instance was constructed by concatenating some number of these 6-bit problems together. The three experiments used 5, 10 and 15 such subproblems, resulting in problem sizes of 30, 60 and 90 bits.

3.8.3. Fully Deceptive

The fully deceptive problems are also representative of a class of problems that have received wide attention in the study of GAs. The particular problem addressed here is from the same source as the fully easy problem, [105], and the problem instances also have 5, 10 and 15 subproblems each of six bits. These problems get their name from reasoning that indicates that they should be difficult for GAs. The low-order schema all lead the GA away from the single global maximum. The subproblem appears in Table 3.

Table 3. A fully deceptive 6-bit problem. Unitation is the number of bits in the binary string that are set to one. The maximum fitness point is the string 111111 with six ones.

Unitation	0	1	2	3	4	5	6
Fitness	0·90	0·45	0·35	0·30	0·30	0·25	1·00

The first thing to notice about this problem is that the location of the global maximum is the string of all ones, the opposite of the fully easy problem. This problem has one global maximum (all ones) and one β-local-maximum (all zeroes). Under steepest ascent, the global maximum can be reached from only seven points in the space. The β-local-maximum will be reached from the rest of the space.

3.8.4. Distributed Fully Deceptive

The distributed deceptive problem contains a number of fully deceptive subproblems, identical to those in the fully deceptive problem above. The difference is that the bits comprising each subproblem are spread through the string. Thus if the problem contains five 6 bit-subproblems, the bits for the first subproblem are located at positions 0, 5, 10, 15, 20 and 25. The bits for the second subproblem are located at positions 1, 6, 11, 16, 21 and 26 and so on.

3.8.5. Busy Beavers

In Rado's "busy beaver" problem [106], deterministic Turing Machines (TMs) with alphabet $\{0, 1\}$ and a fixed number of states k (plus a halt state) are sought that write as many 1 symbols as possible on an initially zero-filled two-way infinite tape before halting. The number of 1 symbols that remain on the tape is the machine's score. The problem is to find, for a given k, a halting TM with the highest score, which will be denoted by $\Sigma(k)$. More than one machine may generate $\Sigma(k)$ ones. The busy beaver problem is extremely difficult and has an interesting history. The problem is solved for $k \leq 4$. There are two 5-state machines that halt and leave 4,098 ones on the tape and a 6-state machine that produces over 95 million! See Brady [107, 108] for details of $k = 4$ and Marxen and Buntrock [109] for $k = 5$. The results of applying a GA to the problem and the original presentation of reverse hillclimbing (the subject of Chapter 4) can be found in [110].

3.8.5.1. Fitness and Representation

The fitness of a TM was its score, or -1 if the machine did not halt. The general halting problem is not an issue for $k \leq 4$ (as in our experiments), since the problem is solved for machines with up to 4 states. In each case, the maximum number of steps taken by a halting machine is known and can be used to terminate evaluation.

A TM with k states was represented by a character string of $6k$ bytes. When the machine is in a certain state, looking at a certain symbol, it needs to know three things: the next state, the symbol to write on the tape and the direction in which to move. Since the machine could be scanning either a 0 or a 1, six bytes are required. By restricting crossover to byte boundaries, only legal TM's were generated.

This representation allows the use of simple operators. There are a number of symmetries that other approaches have taken advantage of that are not exploited here. For example, the direction the tape head moves when the machine enters the halt state makes no difference to the number of ones left on the tape. Noticing this, and including it in the representation halves the size of the search space. Similarly, the symbol written to the tape on the transition into the halt state can be assumed to be a 1 as this assumption cannot reduce the number of ones the machine has written. Other symmetries include the renaming of states, and replacing left moves with right and vice-versa.

The "representation" that has been most widely used in other work on this problem is known as Tree Normal Form. This takes advantage of all the above symmetries. Unfortunately it is not so much a representation as an algorithm for producing and running TMs. If such a representation were adopted, and there is no doubt that it could be, it would also require the construction of specialized operators designed to produce TMs of the correct form. These issues are exactly those that were discussed in §1.3(5). Once again, intelligent observations about the nature of the problem lead to representations that greatly reduce the size of the problem. For example, for 4 states TNF produces approximately 600,000 machines instead of 25,600,000,000.

3.8.5.2. Mesas

The combination of representation and operator here results in a landscape with many mesas. The operator used by BH changes any state into any other, a left movement into a right or a one into a zero. Every halting TM in the landscape is connected by an edge to a twin that produces the same number of ones. This corresponds to changing the direction of movement when entering the halt state. Thus every TM with no fitter neighbors is part of a mesa of size at least 2 which, unless modified, BH can never escape. This was the motivation for imposing a limit on the number of non-improving steps that BH allows.

It is also interesting to note that every halting TM is connected by an edge to at least one TM that does not halt. A halting TM can be made to not halt by modifying the behavior when in the initial state so it remains in the state (either symbol may be written and either direction may be taken). Thus every machine with maximal fitness also has at least one neighbor that has lowest fitness. Similarly, each TM with a single transition into the halt state can be made to not halt by altering this transition. These phenomena are the result of a simplistic choice of representation and operators.

3.8.6. Holland's Royal Road

The Royal Road functions were introduced by Mitchell, Forrest and Holland [36]. They were designed as functions that would be simple for a genetic algorithm to optimize, but difficult for a hillclimber. However, a form of hillclimbing suggested by Palmer [111], easily outperformed a genetic algorithm on what had been expected to be the simplest Royal Road function for that algorithm. Recently, Holland [112] presented a revised class of Royal Road functions that were designed to create insurmountable difficulties for a wider class of hillclimbers, and yet still be admissible to optimization by a genetic algorithm. There is no easily understandable presentation of these functions available elsewhere. This section provides one and is somewhat lengthy as a result.

Holland used seven variables in the description he presented to the genetic algorithms community. In describing the class of functions, I will use his variable names where reasonable, and indicate deviations. Holland suggested a default value for each of these variables and in what follows I will use these values to provide illustration. The following terms will be used consistently in what follows: *block*, *complete*, *gap*, and *region*. These are introduced below and can be relied on to always mean exactly the same thing.

3.8.6.1. Description

The function takes a binary string as input and produces a real value which the searcher must maximize. The string is composed of 2^k non-overlapping contiguous regions, each of length $b + g$. With Holland's defaults, $k = 4$, $b = 8$, $g = 7$, there are 16 regions of length 15, giving an overall string length of 240. We will number the regions, from the left, as $0, 1, \ldots, 2^k - 1$.

Each region is divided into two non-overlapping pieces. The first, of length b, will be called the block. The second, of length g, will be called the gap.[3] In the fitness calculation, only the bits in the block part of each region are considered. The bits in the gap part of each region are completely ignored during fitness calculation. Holland's description called the block part of each region by various names: "building blocks," "elementary building blocks," "schemata," "lower level target schemata," and "elementary (lowest-level) building blocks (schemata)."

The fitness calculation proceeds in two steps. Firstly, there is what Holland calls the PART calculation. Then follows the BONUS calculation. The overall fitness assigned to the string is the sum of these two calculations.

The PART Calculation

The PART calculation considers each block individually, and each in exactly the same fashion. Each block receives some fitness score, and these are added to produce the total PART contribution to the overall fitness. The fitness of each block is based entirely on the number of 1 bits it contains.[4] The idea is to reward 1's up to a certain point. Every 1 up to (and including) a limit of m^* adds to the block's fitness by v. Thus, with the default settings, a block with three 1's would have fitness $3 \times 0{\cdot}02 = 0{\cdot}06$. If a block contains more than m^* 1's, but less than b 1's, it receives $-v$ for each 1 over the limit. With the default settings $m^* = 4$, and so a block with six 1's is assigned a fitness of $(6 - 4) \times -0{\cdot}02 = -0{\cdot}04$.

Finally, if a block consists entirely of 1's (i.e., it has b 1's), it receives *nothing* from the PART calculation. Such a block will be rewarded in the BONUS calculation. If a block consists entirely of 1 bits, it will be said to be *complete*. From the above, we can construct a table of fitness values based on the number of ones in a block.

---

[3]Holland did not give a name for this part of the regions, or give a variable name for its length. These gaps have been called "introns" (borrowing from biology) and have been used with varying degrees of success [96, 113] to alter the effects of crossover in genetic algorithms.

[4]Holland used $m(i)$ to denote the number of 1's in block i. I will not use a variable.

Table 4 gives the values for the default settings.

Table 4. PART fitness values for Holland's default settings. Unitation is the number of bits in the binary string that are set to one. Although the $\binom{8}{4}$ strings with unitation 4 have the highest PART fitness, these strings are not optimal in the complete function as the BONUS calculation assigns a fitness of 1·0 to the string 11111111.

Unitation	0	1	2	3	4	5	6	7	8
Block fitness	0·00	0·02	0·04	0·06	0·08	−0·02	−0·04	−0·06	0·00

The BONUS Calculation

The idea of the BONUS calculation is to reward completed blocks and some combinations of completed blocks. Holland gives rewards for attaining what he calls "levels." At the lowest level, zero, rewards are given for complete blocks (i.e., blocks that consist entirely of 1's).[5] If such a block exists, it receives u^* as its fitness. Any additional complete blocks receive a fitness of u. Thus, with the default settings, a string that contained three complete blocks would receive a BONUS fitness of $1{\cdot}0 + (2 \times 0{\cdot}3) = 1{\cdot}6$ for level 0.

But that is not the end of the BONUS story. At the next level, *pairs* of blocks are rewarded. Suppose we label the blocks from left to right as $B_0, B_1, \ldots, B_{2^k-1}$. The function then rewards any completed pair (B_{2i}, B_{2i+1}) for $0 \leq i < 2^{k-1}$. The rewards are calculated in the same way as they were done at level 0. The first completed pair of blocks receives u^*, and additional completed pairs receive u. Notice that not all pairs of completed blocks are so rewarded. Only those contiguous pairs whose indices are of the form $(2i, 2i+1)$.

We can now calculate rewards for the next level, that of quadruplets of completed blocks, and this is done in the same fashion. In general, there are $k+1$ levels, and at level $0 \leq l \leq k$, we are looking for contiguous sets of 2^l complete blocks, whose first block is labeled B_{i2^l} with $0 \leq i < 2^{k-l}$. At all levels, the first such set of complete blocks receives fitness u^*, and additional sets of completed blocks receive fitness u. The total fitness for the level is the sum of these fitnesses.

[5]Holland numbered his lowest level 1 not 0.

To make this more concrete, consider the function with Holland's default values. With $k = 4$ we have 16 regions and each contains a block of length $b = 8$. The BONUS fitness calculation rewards completed single blocks (this is level 0), and rewards the completion of the sets $\{B_0, B_1\}$, $\{B_2, B_3\}$, $\{B_4, B_5\}$, $\{B_6, B_7\}$, $\{B_8, B_9\}$, $\{B_{10}, B_{11}\}$, $\{B_{12}, B_{13}\}$, $\{B_{14}, B_{15}\}$, (level 1) $\{B_0, \ldots, B_3\}$, $\{B_4, \ldots, B_7\}$, $\{B_8, \ldots, B_{11}\}$, $\{B_{12}, \ldots, B_{15}\}$ (level 2) $\{B_0, \ldots, B_7\}$, $\{B_8, \ldots, B_{15}\}$ (level 3) and finally $\{B_0, \ldots, B_{15}\}$ (level 4) of completed blocks. The total BONUS contribution to the fitness is computed by adding the fitness at each of the $k + 1$ levels.

3.8.6.2. Experiments

The experiments on Holland's royal road in this chapter only varied k, which determines the number of level zero blocks and hence the number of levels. The three values used were 2, 4 and 6, giving problems with 60, 240 and 960 bits.

3.9. Results

The results of twelve of the eighteen experiments are shown in Figures 13 to 24. Results for the smallest instance of each problem are not shown. Performance on the smallest instances of each problem qualitatively matched performance on the larger instances. In all these graphs, the X axis (mean number of evaluations) has a log scale. Each of the algorithms was run at least 2000 times on each instance of each problem and in most cases at least 10,000. Each time an algorithm achieves a performance level (for instance, completing a fully deceptive subproblem), the number of evaluations is recorded. From this, the mean number of evaluations taken to achieve performance levels is plotted for each algorithm on each problem. The line representing an algorithm is terminated at the highest level that was achieved at least ten times. Standard errors for the following graphs are presented in Tables 40 to 69 in Appendix C.

3.9.1. Overall Trends

A quick perusal of the result graphs shows that the CH algorithm has performed remarkably well. On these experiments it is certainly the most consistent algorithm—in no experiment does it have the worst performance. The hillclimber is clearly the worst algorithm on four of the functions (fully deceptive, distributed fully deceptive, fully easy and Holland's royal road), and the two GA variants are clearly the worst algorithms on the remaining two (busy beavers and one max). On three of the functions (fully deceptive, fully easy and Holland's royal road), the CH algorithm is the best performer and it can easily be argued that it is the best on a fourth (busy beavers). Only on one problem of the six, one max, is it significantly beaten (by BH, which is hardly surprising).

It is interesting to note how much consistency there is in the graphs for each experiment. Each of the six experiments has three graphs and within each set of three there is remarkable consistency. The shapes of the lines representing each algorithm are well preserved over the graphs, as are the relative orderings of the algorithms and the ways that the orderings change, and when they change, as higher performance levels are achieved.

The addition of elitism to the GA resulted in improvement in every case (though some were very minimal). However, in no case was the improvement great enough to significantly alter the relative ranking of the GA without elitism. For example, if the order of the three algorithms CH, GA-S and BH was, say, BH, GA-S, CH from best to worst, then the performance of GA-E would fall between that of BH and GA-S.

Table 5 gives a rough overview of the performance of the algorithms. Each

Table 5. Overall trends in the crossover hillclimbing experiment. Algorithms are roughly ranked according to performance on the six problems. 1 = best, 3 = worst. The standard and elitist versions of the GA are represented as a single column.

	BH	CH	GA
Busy beavers	1	2	3
One max	1	2	3
Fully deceptive	3	1	2
Distributed deceptive	3	1	1
Fully easy	3	1	2
Holland's royal road	3	1	2

algorithm is ranked for each problem, with a rank of 1 being the best. When it was not clear which algorithm was better from looking at the three experiments for a problem, preference was given to the algorithm that performed best on the largest instance of the problem and algorithms received equal rank if there was still no clear difference. The two versions of the GA have been coalesced since they virtually always rank next to each other. This allows a clearer view of the overall pattern. It should be clear that CH is never the worst, the GA is never clearly the best and BH is either best or worst.

3.9.2. One Max

Figures 13 and 14 show the results of the four algorithms on the one max problem

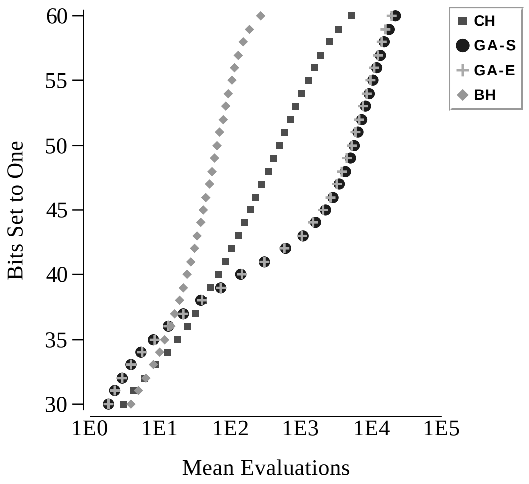

Figure 13. Mean evaluations to solve a 60-bit one max problem using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

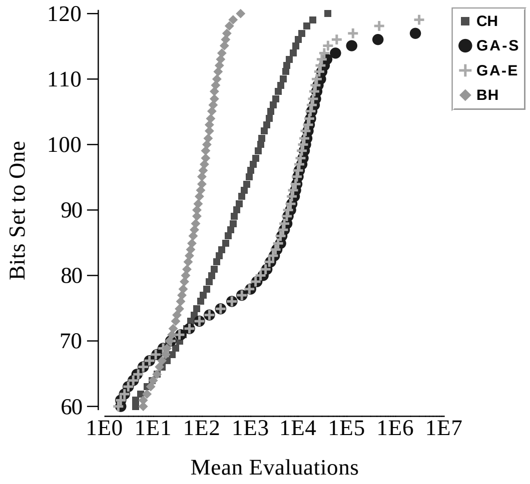

Figure 14. Mean evaluations to solve a 120-bit one max problem using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

described in §3.8.1(58). These are consistent with what we might expect from previous work by Ackley [26] and Culberson [29]. Ackley's results show two versions of a hillclimber clearly outperforming two versions of a GA. Culberson demonstrated the existence of $\mathcal{X}_1^{2\to2}$-local-maxima in the component of the one-point crossover landscape that corresponds to complementary binary strings for the one max function. There are $\mathcal{X}_2^{2\to2}$-mesas in components of the two-point crossover landscape for non-complementary strings. A trivial example is any pair (a, a) where a is not the global maximum. In general, if $f(a, b) = \max\{g(a), g(b)\}$ and $g(v)$ is the number of ones in the string v, then (a, b) will be a $\mathcal{X}$-local-maximum if b has a 0 everywhere a does and a is not the global maximum (or vice-versa). For example, $(1101101, 1001001)$ is a $\mathcal{X}$-local-maximum since no crossover (of any standard type) can produce a pair containing a member with more than five 1s.

Thus the crossover landscape contains $\chi_2^{2\to2}$-mesas and $\chi_2^{2\to2}$-local-optima, and these will naturally degrade the performance of a simplistic hillclimbing navigation strategy that uses $\chi_2^{2\to2}$. By choosing a new partner for the best string of the current pair, these can be escaped, but this is only performed a fixed number of times. As the algorithm comes closer to locating the global maximum, the probability of jumping to a $\chi_2^{2\to2}$-local-maximum by choosing a new partner increases. Of course, a vertex may not be a $\chi_2^{2\to2}$-local-maximum but because the navigation strategy of CH only examines a small number of neighbors (10), it may not find the crossover that would produce the fitness increase.

The BH algorithm has a very similar navigation strategy but the landscape it is operating on has no β-local-maxima and so it can never fail to find the global maxima if it is run sufficiently long. Also, this algorithm examines as many as twice the number of neighbors while looking for an uphill direction, so it is unlikely to miss an opportunity to increase fitness when this is possible.

3.9.3. Fully Easy

Figures 15 and 16 show the results of the four algorithms on the fully easy problem

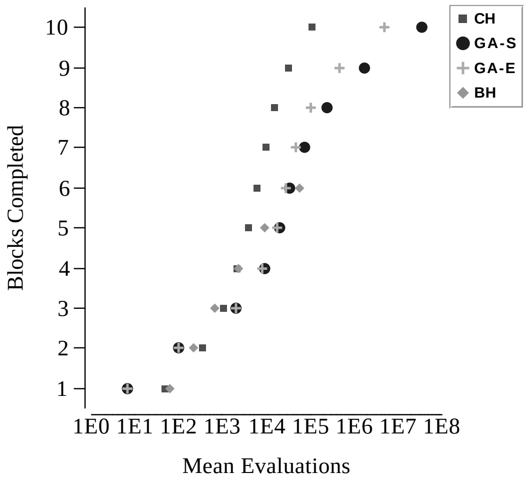

Figure 15. Mean evaluations to solve 10 fully easy 6-bit problems using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

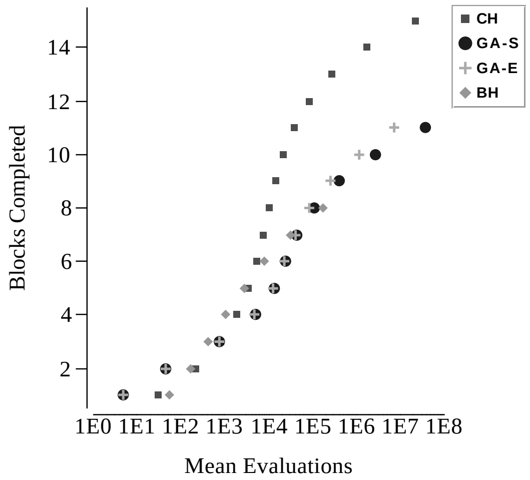

Figure 16. Mean evaluations to solve 15 fully easy-6 bit problems using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

described in §3.8.2(59). The most striking feature of these graphs is the vastly superior performance of the CH algorithm on the largest of these problems (15 fully easy subproblems of length 6). Not only does the CH algorithm maximize all 15 blocks, it does so using less evaluations on average than GA-S uses to maximize 11 of them (the highest level GA-S achieves). The CH algorithm achieves that level (11 blocks), using an average of almost 1000 times fewer evaluations than GA-S. GA-E does better, but CH is still using two orders of magnitude fewer evaluations at the highest level achieved by GA-E.

The performance of BH on this function is particularly poor. As discussed in §3.8.2(59), each subproblem can only be solved by steepest ascent hillclimbing (using β) from 7 of the 64 possible starting points. The situation is not much better for the BH algorithm. In addition to those seven starting points (from which it will find the maximum with probability 1 if left to run long enough), there are $\binom{6}{2} = 15$ strings (with 2 ones) from which the BH will find the global maximum with probability one-third. Thus BH can be expected to solve each fully easy subproblem with a probability of only 12/64.

Even if the landscapes generated by mutation and crossover were equal in terms of numbers of peaks and the sizes of their basins of attractions (from the point of view of the hillclimbing algorithms involved), CH would have an advantage over BH. This stems from the navigation strategy it employs. The CH algorithm operates on a pair of points, and thus it is not necessary to consider a hillclimb finished just because the current pair is not making progress. The worse of the two is discarded and replaced with a random new partner which may take the search to another component of the crossover landscape and perhaps away from a χ-local-maximum. It would perhaps be fairer to allow the BH algorithm to do approximately the same thing and randomly set some fraction of its bits when what appears to be a β-peak is reached. This algorithm was used by Palmer and Pond, who found that replacing approximately a quarter of the bits in the current individual worked best on their problems [114].

3.9.4. Fully Deceptive

Figures 17 and 18 show the results of the four algorithms on the fully deceptive problem described in §3.8.3(59). The results are very similar to the results for the fully easy problem of §3.9.3(68), though with less disparity between the CH algorithm and the GA. Once again, CH performs very well, requiring approximately ten times fewer evaluations than GA-S to achieve the highest performance levels. The performance of GA-E was quite close to that of the CH algorithm on the largest of the problems. No algorithm managed to optimize all 15 deceptive blocks in the hardest experiment. BH performed very poorly.

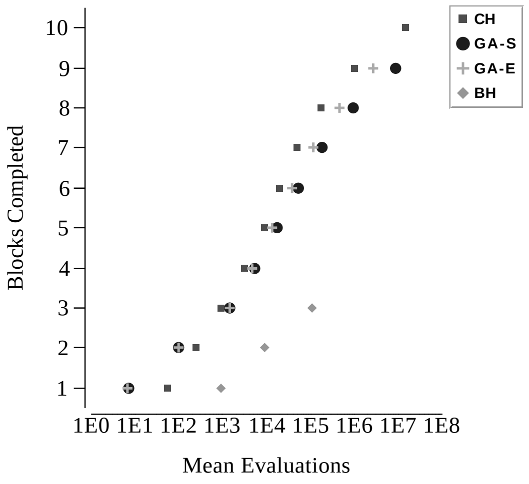

Figure 17. Mean evaluations to solve 10 fully deceptive 6-bit problems with crossover hill-climbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hill-climbing (BH).

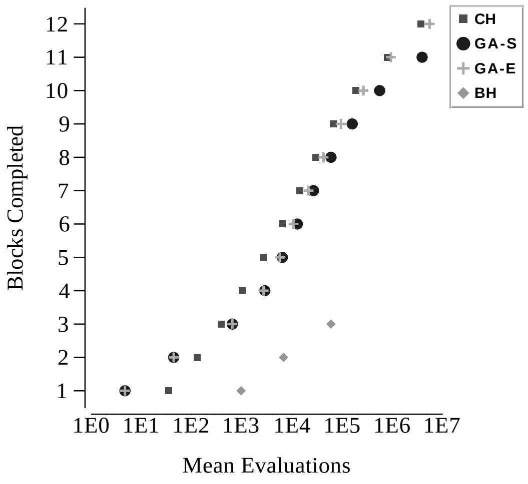

Figure 18. Mean evaluations to solve 15 fully deceptive 6-bit problems with crossover hill-climbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hill-climbing (BH).

3.9.5. Distributed Fully Deceptive

Figures 19 and 20 show the results of the four algorithms on the distributed fully deceptive problem described in §3.8.4(60). CH and the GA-E perform the best on this function. Because their performances are so close, they are both given a 1 ranking in Table 5 on page 66. It is interesting to note that the crossover-based algorithms do better than the bit-flipping based BH, despite the representation being very poor for crossover. The performance of BH is virtually identical to that of the fully deceptive problems, which is as it should be since the distributed representation will not affect this algorithm.

3.9.6. Busy Beavers

Figures 21 and 22 show the results of the four algorithms on the busy beaver problem described in §3.8.5(60). In previous work (with G. Rawlins) I compared the performance of a GA with that of a hillclimber on this problem [110]. The results obtained there were for a slightly different hillclimbing algorithm (steepest ascent), but qualitatively are in agreement with the relationship between the GA and BH found here.

The BH algorithm performs better than the GA or the CH algorithm for the 2-

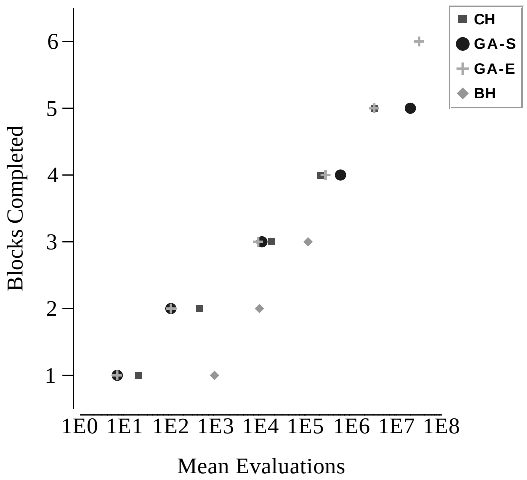

Figure 19. Mean evaluations to solve 10 distributed fully deceptive 6-bit problems using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

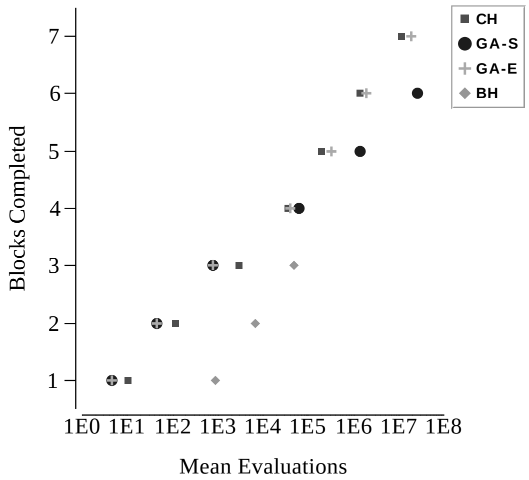

Figure 20. Mean evaluations to solve 15 distributed fully deceptive 6-bit problems using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

and 3- state problems. The 4-state problem is considerably more difficult than either of these and here the CH found an optimal TM (producing 13 ones) several times whereas none of the other algorithms did. To find a machine producing 12 ones, BH took on average approximately 7 million evaluations while the CH algorithm took approximately 8 million. GA-S took approximately 45 million and GA-E 14 million. In all cases, the GA performed worse than either of the other algorithms, though the addition of elitism reduced the gap considerably. The performance of BH and CH was very close on the two larger experiments.

3.9.7. Holland's Royal Road

Figures 23 and 24 show the results of the four algorithms on Holland's royal road function that was described in §3.8.6(62). The CH algorithm is a clear winner for the $k = 2$ and $k = 4$ experiments (with 60 bits and 240 bits respectively). On the final experiment, the performance of the two GAs and the CH algorithm are virtually indistinguishable, with the GAs coming out slightly ahead. In all cases, the BH algorithm does very poorly, which is hardly surprising as one of Holland's design goals was to create a function that would be very difficult to optimize via mutation-based hillclimbing.

The closeness in the results for the GAs and the CH algorithm may be due to

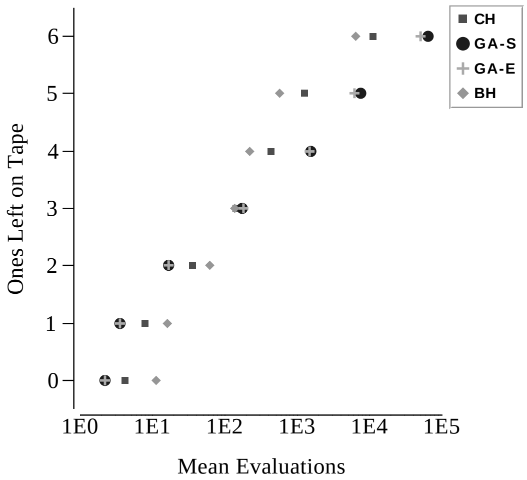

Figure 21. Mean evaluations to find 3-state Turing machines for the busy beaver problem using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

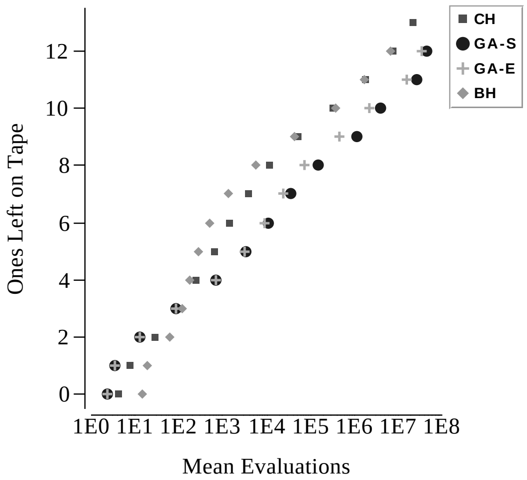

Figure 22. Mean evaluations to find 4-state Turing machines for the busy beaver problem using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

the comparatively large size of the problem (960 bits). It is worth noting that the algorithms actually climb *fewer* levels (only 3) in this problem than in the simpler $k = 4$ problem (where they climb 4). The harder problem has 7 levels that could have been climbed whereas the easier problem only has 5. Increasing the parameters on the GAs and CH to allow them to search longer may have revealed a difference.

3.10. Why Does CH Perform so Well?

It is clear that the CH algorithm performs very well on the problems considered. Rather than examine the relative performance of the four algorithms in detail, we will concentrate on the CH algorithm and attempt to discover why it performs so well.

A first impression of the above results might be that the CH algorithm isolates crossover and that it provides a clear demonstration that crossover is a powerful operator, at times more powerful than mutation. The algorithm was designed to examine crossover in a simple context. To a certain extent this has been achieved, but one loose end remains to be considered. CH is creating new individuals in two ways: through the use of crossover and at random. Initially, two random individuals are created and thereafter a random individual is created each time the algorithm

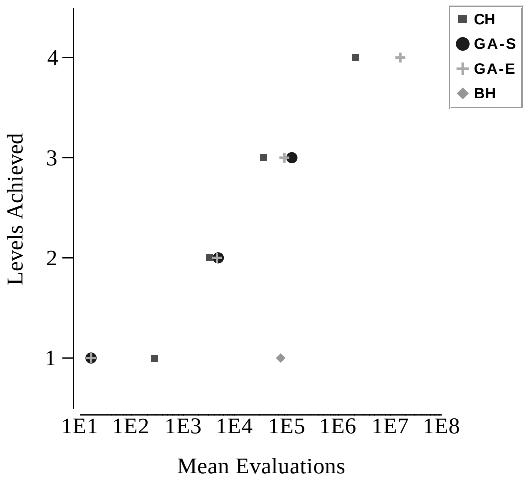

Figure 23. Mean evaluations to climb levels in Holland's royal road with $k = 4$ using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

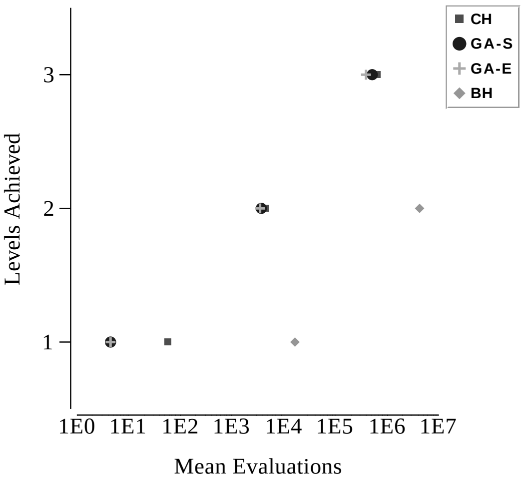

Figure 24. Mean evaluations to climb levels in Holland's royal road with $k = 6$ using crossover hillclimbing (CH), a standard GA (GA-S), an elitist GA (GA-E) and random mutation hillclimbing (BH).

jumps to a new hypercube. How important are these random individuals, and what role do they play in aiding the search? It is simple to examine the number of times a random individual is fitter than any individual seen so far. When this is done, the results are unsurprising. Random creation produces very few highly fit individuals. For example, Table 6 shows the number of times a randomly created individual was responsible for an increase in best fitness over 1000 runs of CH on the 4-state busy beaver problem. In this example, a Turing machine that left seven ones on the tape before halting was discovered on 990 of the 1000 runs, but on only 1 of these was the first such discovery a result of random creation.

To get an idea of the importance of the random individuals as a source of material for crossover to exploit, the CH algorithm used above, CH(10,3,1000), was compared to two variant crossover hillclimbing algorithms, CH(10,3000,0) and CH(10,1,3000). The two new algorithms represent extremes of the spectrum with respect to the random creation of new individuals. CH(10,3000,0) never jumps to a new hypercube when no acceptable crossover can be found from the current pair. Each search will therefore take place entirely within the hypercube determined by the initial choice of random individuals. When an improving (or equaling) crossover cannot be found from the current pair, the search is concluded. This algorithm will be denoted by CH-NJ (NJ = No Jumps). At the other extreme, CH(10,1,3000) allows at most 1

Table 6. The frequency of fitness increases due to random discovery in 1000 runs of the CH algorithm on the 4-state busy beaver problem. The new individual must be the best found so far to gain mention here. The number of times each fitness level was discovered is shown for comparison.

Fitness level	Random discovery	Overall discovery	Fitness level	Random discovery	Overall discovery
0	45	1000	7	1	990
1	133	1000	8	0	786
2	191	1000	9	0	307
3	69	1000	10	0	60
4	24	1000	11	0	13
5	5	1000	12	0	4
6	0	1000	13	0	2

step to be taken by a pair within a hypercube before a jump to another hypercube is forced. This algorithm will be denoted by CH-1S, (1S = 1 Step). These parameter settings limit all three of these algorithms to 3000 steps via crossover.

Figures 25 to 30 show the results on the largest instance of each of the problems (with the exception of Holland's royal road, which shows the default problem). It is clear that CH-1S is the best of the three algorithms on these instances of these problems. Experiments with other settings of the three parameters of the crossover hillclimbing algorithm (not shown) have provided further evidence that the algorithm performs better the more often it jumps between hypercubes. CH-1S is extreme in this respect, as it jumps after every single crossover. This is remarkable as the first crossover performed after a jump always involves a random individual (the jump is performed by replacing one of the current pair by a random individual). As will be seen, crossover involving an individual v and a random individual can be seen as a macromutation of v. For this reason, the CH-1S algorithm is simply performing macromutations that are distributed on the genome in the fashion of two-point crossover.

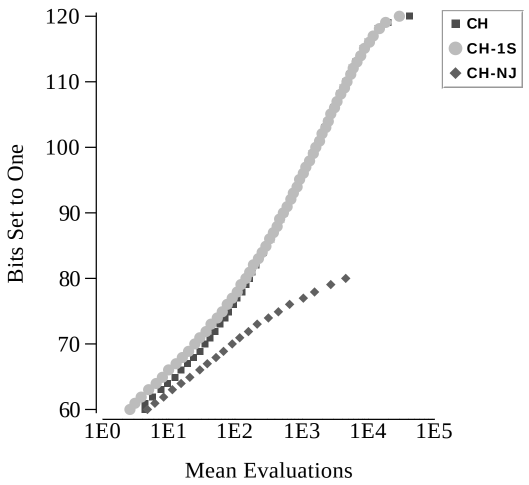

Figure 25. Mean evaluations to solve a 120-bit one max problem using crossover hillclimbing (CH), crossover hillclimbing with no jumps between hypercubes (CH-NJ) and crossover hillclimbing with at most one step per hypercube (CH-1S).

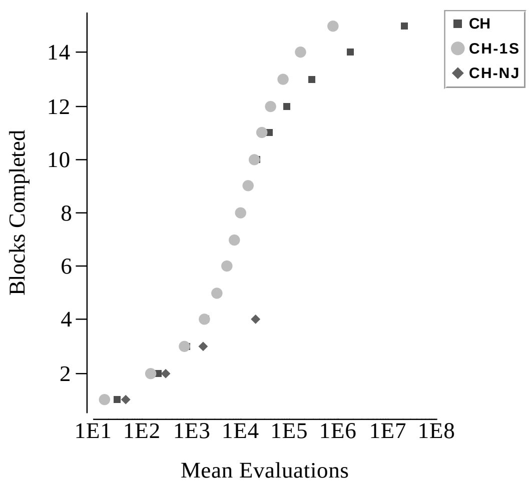

Figure 26. Mean evaluations to complete 15 6-bit fully easy subproblems using crossover hillclimbing (CH), crossover hillclimbing with no jumps between hypercubes (CH-NJ) and crossover hillclimbing with at most one step per hypercube (CH-1S).

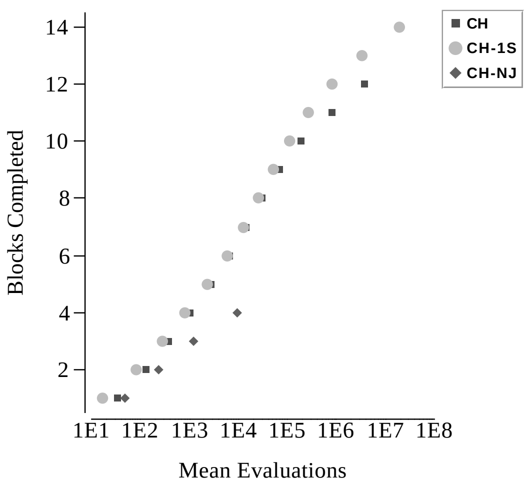

Figure 27. Mean evaluations to complete 15 6-bit fully deceptive subproblems using crossover hillclimbing (CH), crossover hillclimbing with no jumps between hypercubes (CH-NJ) and crossover hillclimbing with at most one step per hypercube (CH-1S).

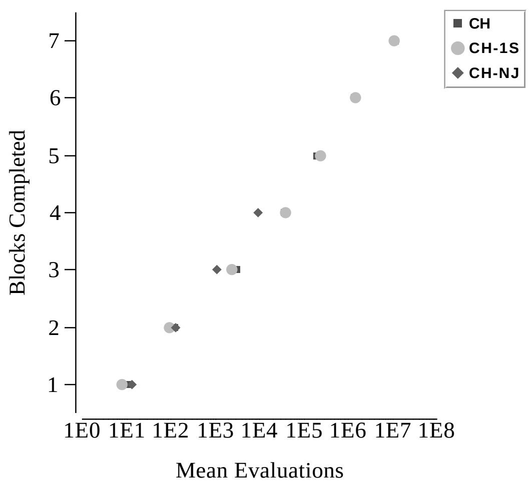

Figure 28. Mean evaluations to complete 15 6-bit distributed fully deceptive subproblems using crossover hillclimbing (CH), crossover hillclimbing with no jumps between hypercubes (CH-NJ) and crossover hillclimbing with at most one step per hypercube (CH-1S).

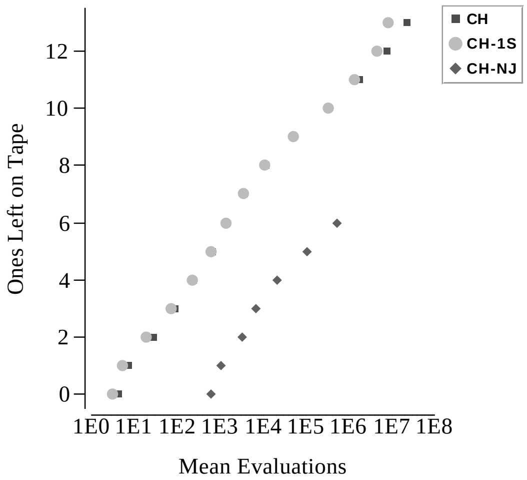

Figure 29. Mean evaluations to find 4-state busy beaver TMs using crossover hillclimbing (CH), crossover hillclimbing with no jumps between hypercubes (CH-NJ) and crossover hillclimbing with at most one step per hypercube (CH-1S).

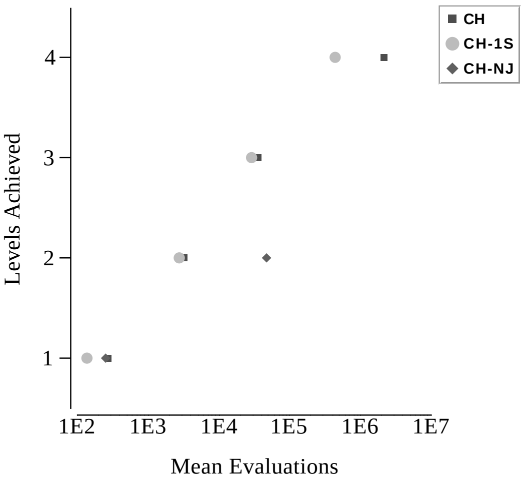

Figure 30. Mean evaluations to achieve levels on Holland's royal road with $k = 4$ using crossover hillclimbing (CH), crossover hillclimbing with no jumps between hypercubes (CH-NJ) and crossover hillclimbing with at most one step per hypercube (CH-1S).

3.11. Crossover: The Idea and The Mechanics

It is important to consider crossover as playing two roles. These can be called the *idea* and the *mechanics* of crossover. The idea of crossover is that there is some reason to believe that each of the parents has a better than average chance of bestowing better than average genetic material upon its offspring. In a GA, the initial population is simply a collection of random individuals, but soon after this, under the influence of selection, the population is made up of individuals whose performance is better than random. The main point of keeping a population and using crossover is to take better individuals and combine them to potentially form even better individuals. The idea of crossover is clearly present in Holland's early work [102, pages 140, 167 and 180]. The mechanics of crossover is the process by which an attempt is made to implement the idea, for instance via one-point, two-point, uniform or one of the many forms of crossover used for representations other than fixed-length strings. All forms of crossover share a similar idea, but the mechanics vary considerably.

The experiments with various forms of crossover hillclimbing show that crossover may be very useful even in the absence of the idea of crossover. That is, even when there is no reason to believe that *both* parents have an above average chance of contributing above average material to the offspring, crossover may still be useful simply through performing macromutation via its mechanics. When considering whether crossover is useful to a GA, we should attempt to distinguish the gains the algorithm is making using the idea of crossover from those made simply through the mechanics. If there is no additional gain due to the idea of crossover, it may be the case that we would do as well if we discarded the population (and thus the GA) and instead used an algorithm that employed macromutation.

3.12. The Headless Chicken Test

This section presents a simple experiment designed to test whether a GA is making progress via the idea of crossover over that which could be made by simply performing macromutations. To do this, compare the GA with an identical GA that uses *random crossover*. The random crossover operator produces two children, C_1 and C_2, from two parents, P_1 and P_2. To do this it generates two random individuals, R_1 and R_2. The operator then performs a crossover between P_1 and R_1 and keeps one of the children (selected at random), C_1. Then C_2 is produced in an identical fashion, using P_2 and R_2 as parents. The two children are then returned to the GA. This is the only difference between the two algorithms. The random crossover operator is illustrated in Figure 31.

In the algorithm with random crossover, there is no communication between population members since there is no crossover between population members unless, by chance, a randomly generated parent also happens to be present in the population. When one of the individuals involved in a crossover is randomly chosen, the operation is purely mechanical and has nothing of the idea, or spirit, of crossover. In this case, despite the identical mechanical rearrangement, the operation is not crossover. That is, without the idea of crossover, you do not have crossover. To call this operation crossover is similar to calling a headless chicken running around the yard a chicken. Certainly it has many characteristics of a chicken, but an important piece is missing. A further argument that this operation is not really crossover is that two parents are not required. For example, for two-point crossover on strings, simply choose the crossover points and set all the loci between the points to randomly chosen alleles. This is an identical operation and is clearly just a macromutation.

This comparison was performed (1) between GA-S and an identical GA with random crossover (GA-RC) and (2) between GA-E and an elitist GA with random crossover (GA-RCE). Figures 32 to 37 show the result of this experiment on the problem instances used to compare the CH variants. The results of the comparison between GA-E and GA-RCE are not shown (these exhibit identical qualitative results

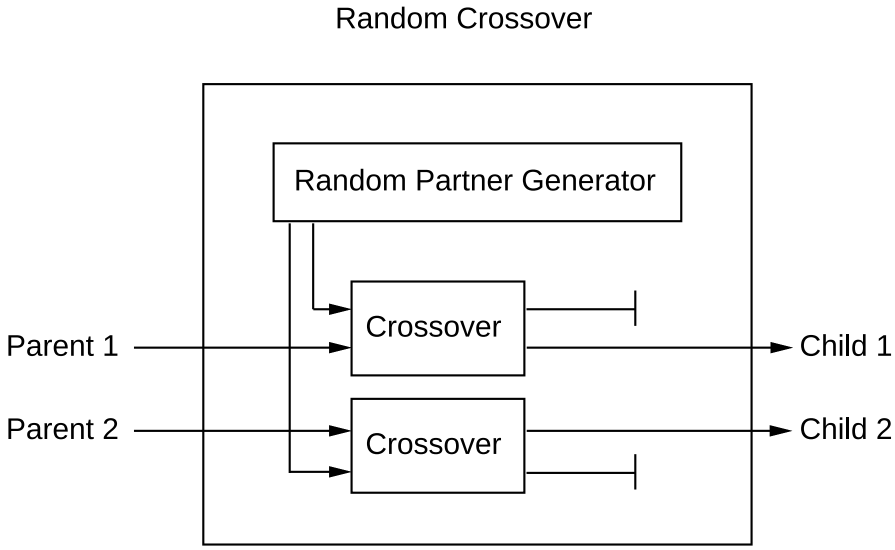

Figure 31. The random crossover operator used in the headless chicken test. When the GA with random crossover passes two parents to the operator, instead of recombining them as a normal crossover would do, two random individuals are created and these are crossed with the parents. One offspring is kept from each of these crosses and these are handed back to the GA. The crossover method used (one-point, two-point, uniform, etc.), is the same as that used in the version of the GA that is being compared to the GA with random crossover.

to those of GA-S versus GA-RC). On all instances of all problems, GA-S is a clear winner over GA-RC when the problem contains well-defined building blocks. This is the case with the fully deceptive, fully easy, Holland's royal road and one max problems. On these problems, the idea of crossover is aiding the search, as we would hope.

More interesting is the comparison on the problems where well-defined building blocks do not exist: the busy beaver and distributed fully deceptive problems. On the busy beaver problems, the performance of the GA and GA-RC is hard to distinguish. On the distributed fully deceptive problem, GA-RC actually outperforms the GA. In this case, the GA is not only using an operator (two-point crossover) that, in an informal sense, has a bias that is orthogonal to the structure of the encoding, but it is also drawing its samples from a population. GA-RC is using the same operator but has the advantage of being able to explore more widely. When GA-RC is as good or better than the normal GA on a problem, it suggests that the idea of crossover is doing nothing significant for the normal GA. In such cases it seems safe to conclude that the chosen combination of representation, fitness function and crossover operator

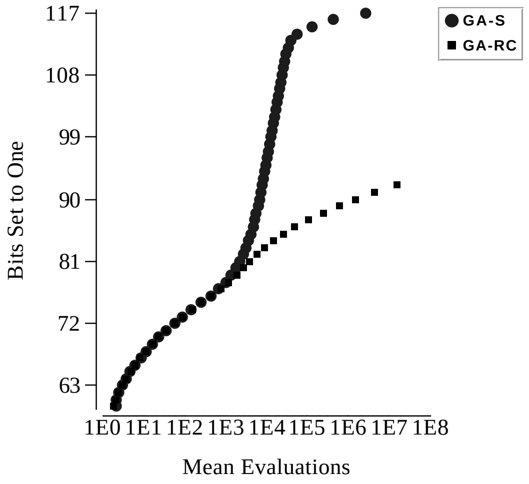

Figure 32. Mean evaluations to solve a 120-bit one max problem using a standard GA (GA-S) and a GA with random crossover (GA-RC).

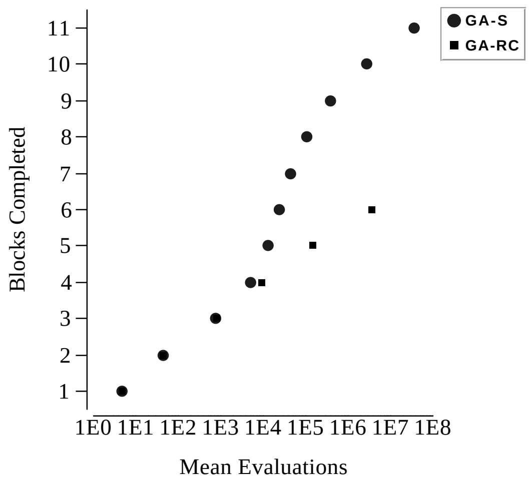

Figure 33. Mean evaluations to complete 15 6-bit fully easy subproblems using a standard GA (GA-S) and a GA with random crossover (GA-RC).

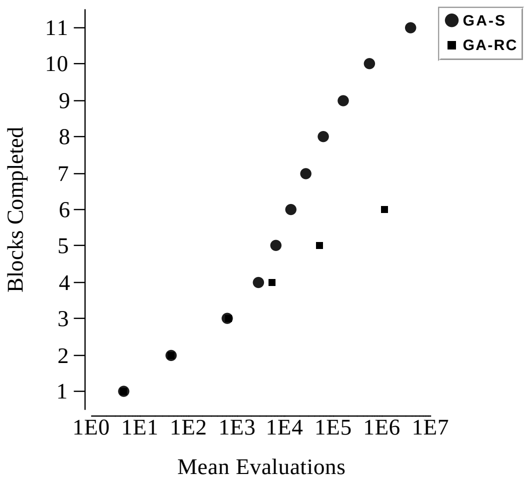

Figure 34. Mean evaluations to complete 15 6-bit fully deceptive subproblems using a standard GA (GA-S) and a GA with random crossover (GA-RC).

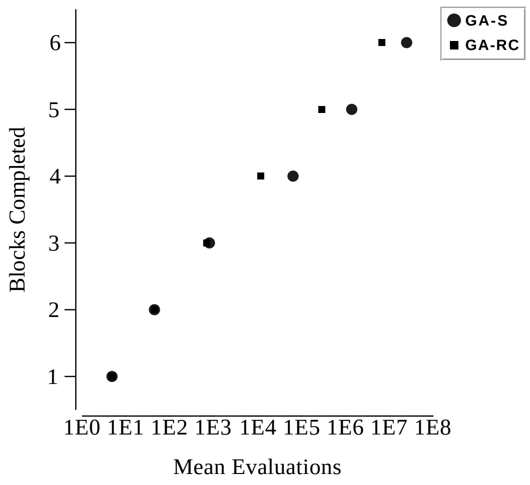

Figure 35. Mean evaluations to complete 15 6-bit distributed fully deceptive subproblems using a standard GA (GA-S) and a GA with random crossover (GA-RC).

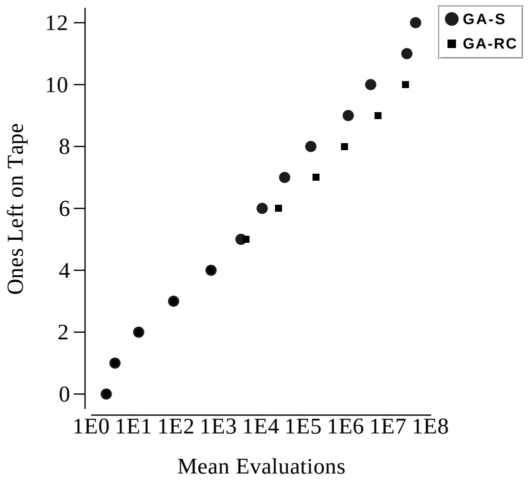

Figure 36. Mean evaluations to find 4-state busy beaver TMs using a standard GA (GA-S) and a GA with random crossover (GA-RC).

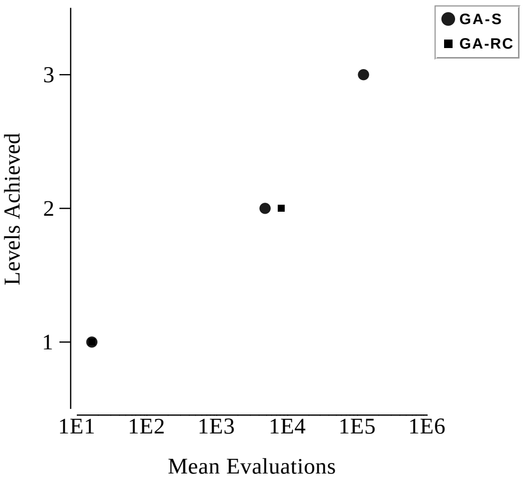

Figure 37. Mean evaluations to achieve levels on Holland's royal road with $k = 4$ using a standard GA (GA-S) and a GA with random crossover (GA-RC).

does not result in a problem that contains building blocks that the GA is managing to exploit using the idea of crossover. De Jong, Spears and Gordon have also identified problems on which the use of crossover results in worse performance [115].

This does not imply that there is no other representation that would contain exploitable building blocks, or that another crossover operator might not do better at exploiting the building blocks that do exist (if any), or that it is not possible to modify the GA in some way so that it does exploit the building blocks (e.g., by using a larger population, a different fitness function, or adopting measures to maintain diversity). This is a conclusion about a single crossover operator, a single representation and a single algorithm, and may well not apply if any of these are altered. The test examines a particular set of choices and nothing more. In particular, the conclusion that a GA is not suitable for a problem, based on a failed headless chicken test, is in no way justified. If a GA fails the headless chicken test (i.e., it does not outperform a GA with random crossover), the GA is not making gains from crossover above those that could be made via explicit macromutation. If this is the case, it is not clear why one would bother to maintain a population or, consequently, use a GA on *this combination* of problem, representation and fitness function.

The experiment suggests an explanation for GAs that appear to make good use of crossover early in a run, but not thereafter. It may be the case that the combination of the representation and fitness function does not have building blocks that can be exploited by the crossover operator in question. In this case, crossover may make improvements simply through macromutations at the start of the run and then, as is widely known, become increasingly less useful as the population converges. Crossover *appears* to be useful (since disabling it results in worse performance), but in fact, its usefulness is merely a consequence of the macromutations it is performing.

3.13. Macromutational Hillclimbing

The success of CH-1S and the observation that it is simply performing macromutation prompted the implementation of two hillclimbers that use a macromutational operator and the navigation strategy of BH. In the mutation operator of the first, (BH-MM), the genome is viewed as a ring. Two distinct points on the ring are selected uniformly at random and the loci in the smaller section are set to random alleles (including those that are already present).

The smaller section is chosen for mutation as observation revealed that CH was making the great majority of its improving steps when it modified less than half the genome. The result of this mutation operator is exactly what a crossover with a random individual would produce if the child that inherited most from the non-random parent was retained. The operator distributes the mutations in the same way that two-point crossover does. This hillclimber is very similar to CH(1,1,∞). In the second hillclimber, the macromutation operator decides on a number of loci to set randomly (as in BH-MM) but then distributes these mutations at random through the string. The behavior of BH-DMM is similar to that which would be obtained via crossover with a random string using uniform crossover.

This operator was described in terms of strings for simplicity, but it is more general. Any representation that can be viewed as containing a number of independently adjustable components could have a similar operator designed for it. Such representations are very common, and were described briefly in §2.6(30).

These two hillclimbers were compared to CH-1S and GA-E. In every case but the busy beaver problems, one of the macromutational hillclimbers always proved the best algorithm. Once again, the results depended on whether well-defined building blocks existed. When they did, BH-MM outperformed BH-DMM by orders of magnitude. When they did not (on the distributed fully deceptive problem), BH-DMM performed an order of magnitude better than BH-MM. On every problem, BH-MM achieved the highest levels of performance at least one order of magnitude faster than GA-E, and in one case (15 fully easy subproblems) it required over 3 orders of magnitude fewer evaluations to achieve the highest level reached by GA-E. Typically, the hillclimbers would also reach levels of performance that were never achieved by the GA. The results for the six problems instances are shown in Figures 38 to 43.

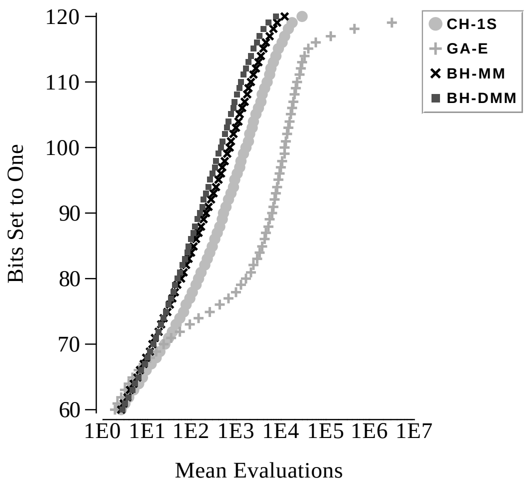

Figure 38. Mean evaluations to solve a 120-bit one max problem using crossover hillclimbing with at most one step per hypercube (CH-1S), an elitist GA (GA-E), hillclimbing with macromutations (BH-MM) and hillclimbing with distributed macromutations (BH-DMM).

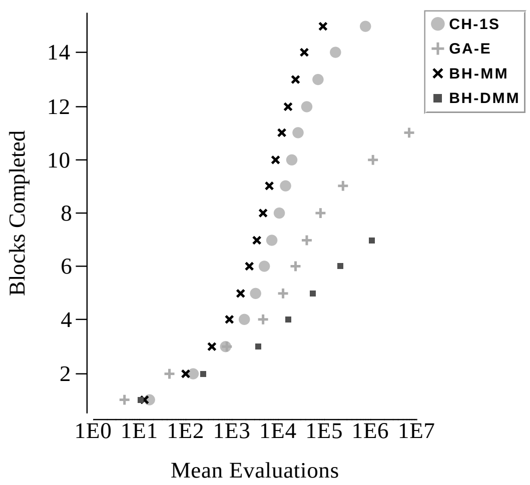

Figure 39. Mean evaluations to complete 15 6-bit fully easy subproblems using crossover hillclimbing with at most one step per hypercube (CH-1S), an elitist GA (GA-E), hillclimbing with macromutations (BH-MM) and hillclimbing with distributed macromutations (BH-DMM).

3.14. Summary

The traditional method used to assess whether crossover is useful to a GA is to compare the GA with crossover to a GA without crossover. This chapter argues that if the GA with crossover performs better, the conclusion that crossover is therefore useful (i.e., that it is facilitating the exchange of building blocks between individuals) is not justified. A more informative method of assessing the worth of crossover to a GA was presented. This compares the normal GA with a GA that uses random crossover. Such a GA dispenses with the idea of crossover while maintaining the mechanics. This makes it possible to see what advantage the standard GA is gaining from the idea of crossover over and above what it could be gaining from the mechanics alone. In some cases, when well-defined building blocks are not present, the GA may actually perform *worse* with normal crossover than a GA with random crossover as a result of its use of a population. This test gives an indication of when building

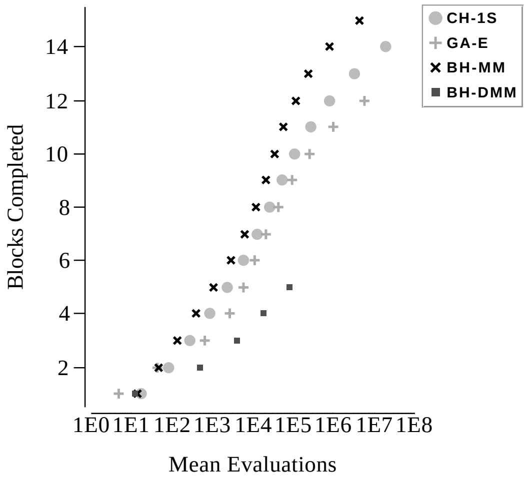

Figure 40. Mean evaluations to complete 15 6-bit fully deceptive subproblems using crossover hillclimbing with at most one step per hypercube (CH-1S), an elitist GA (GA-E), hillclimbing with macromutations (BH-MM) and hillclimbing with distributed macromutations (BH-DMM).

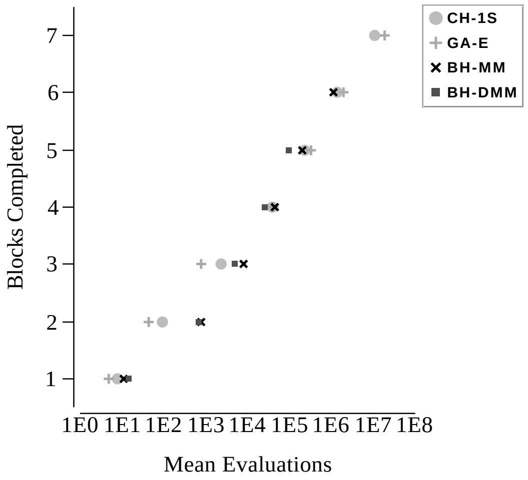

Figure 41. Mean evaluations to complete 15 6-bit distributed fully deceptive subproblems using crossover hillclimbing with at most one step per hypercube (CH-1S), an elitist GA (GA-E), hillclimbing with macromutations (BH-MM) and hillclimbing with distributed macromutations (BH-DMM).

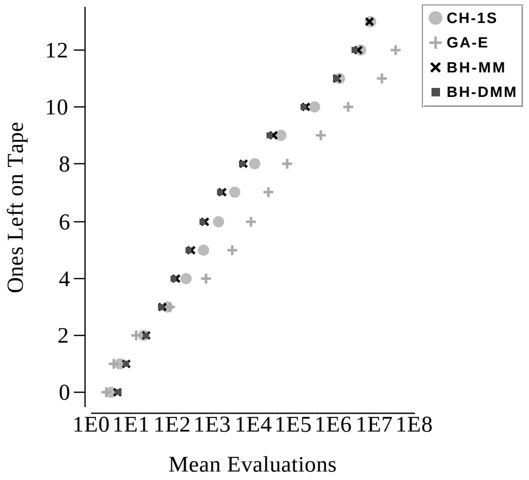

Figure 42. Mean evaluations to find 4-state busy beaver TMs using crossover hillclimbing with at most one step per hypercube (CH-1S), an elitist GA (GA-E), hillclimbing with macromutations (BH-MM) and hillclimbing with distributed macromutations (BH-DMM).

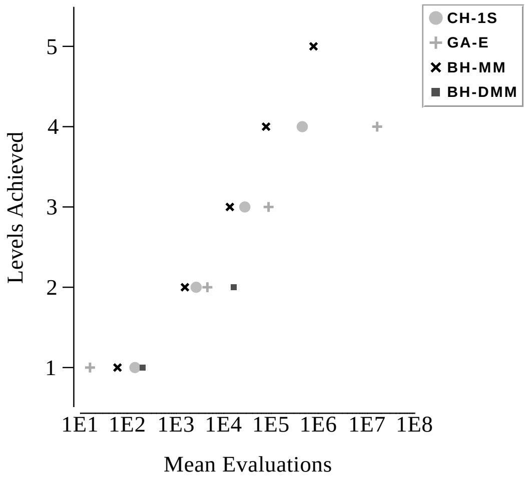

Figure 43. Mean evaluations to achieve levels on Holland's royal road with $k = 4$ using crossover hillclimbing with at most one step per hypercube (CH-1S), an elitist GA (GA-E), hillclimbing with macromutations (BH-MM) and hillclimbing with distributed macromutations (BH-DMM).

blocks (that are exploitable by a GA with a given crossover operator) exist in a representation.

The major reason for the maintenance of a population in a GA is to allow the communication of information between individuals via crossover. When testing whether it is worth bringing a population and crossover to bear on a problem, the null hypothesis should be that the GA with crossover does not outperform the GA with random crossover, *not* that the GA with crossover outperforms the GA without crossover. If this comparison indicates that the idea of crossover is not producing significant gains, the use of a GA is probably not warranted. In these cases, there are simpler algorithms that use the macromutational mechanics of crossover and do not maintain a population, and these can easily outperform the GA. Several such algorithms, based on crossover hillclimbing or macromutational hillclimbing were presented. This point was also made by Fogel and Atmar [88]. On the other hand, when building blocks do exist, these algorithms (especially those that use macromutation exclusively) still outperformed the GA on virtually every instance of every problem addressed. These results support Eshelman and Schaffer's belief that the "niche" in which crossover gives a GA a competitive advantage may be quite small [89]. Actually, the situation may be worse than they feared, as a macromutational hillclimber easily outperforms a GA on Holland's royal road, which has the properties that Eshelman and Schaffer ascribe to problems residing in crossover's niche. The niche becomes smaller if we insist that the idea of crossover, in addition to the mechanics, be responsible for good performance. In the context of function optimization, it is clear that care must be taken if one intends to make effective use of a population, crossover, and a GA.