4

Taken from
Philip David Brierley – 'Some practical applications of neural networks in the electricity industry'
EngD Thesis – 1998 – Cranfield University - UK

Genetic Inspired Optimisation

4.1 What are Genetic Algorithms?

Genetic algorithms (GAs) are directed random search techniques used to look for parameters that provide a good solution to a problem. Essentially they are nothing more than educated guessing. The 'education' comes from knowing the suitability of previous candidate solutions and the 'guessing' comes from combining the fitter attempts in order to evolve an improved solution.

For example, the back propagation algorithm is a gradient based method for finding a weight set for a MLP that best maps the inputs onto the output, a search that can also be performed by GAs [**7**]. The optimisation problem of interest in this work is finding a schedule for electrically charging storage devices (hot water tanks and storage radiators) over a 24 hour period, given that electricity prices vary half-hourly. A solution is sought that minimises electricity costs whilst satisfying the hot water or thermal comfort requirements.

4.2 How do GAs Work?

The inspiration for GAs came from nature and survival of the fittest. In a population, each individual has a set of characteristics that determine how well suited it is to the environment. Survival of the fittest implies that the 'fitter' individuals are more likely to survive and have a greater chance of passing their 'good' features to the next generation. In sexual reproduction, if the best features of each parent are inherited by their offspring, a new individual will be created that should have an improved probability of survival. This is the process of evolution.

In nature the 'blueprint' of individuals is contained within their DNA. The DNA can be thought of as a string of genes, with each gene or combination of genes representing a particular feature. Reproduction is the 'crossover' of two DNA strings to produce a new blueprint that has genes from both parents. Mutation can also occur where a particular gene is not an exact copy of either parent.

In genetic algorithm terms, a candidate solution is often referred to as a chromosome or string, which is a sequence of encoded numbers. This is commonly referred to as a bit string if the numbers are binary encoded.

The process involved in GA optimisation problems is based on that of natural evolution and broadly works as follows,

1. Randomly generate an initial population of potential solutions.

2. Evaluate the suitability or 'fitness' of each solution.

3. Select two solutions biased in favour of fitness.

4. Crossover the solutions at a random point on the string to produce two new solutions.

5. Mutate the new solutions based on a mutation probability.

6. Goto 2.

4.3 The GA Operators

Selection, crossover and mutation are the basic operators involved in GAs. How these and other factors can affect the operation of GAs will be demonstrated by means of several examples and experimental observations.

Consider the popular board game ‘Mastermind’ where a player has to determine a hidden sequence of colours starting from an initial random guess. This initial guess is scored with a black marker for each colour in the correct position and a white marker for a correct colour but in the wrong position. Further guesses are made and scored until the correct sequence is determined or a given number of attempts have been made. In this game the correct solution evolves from the more suitable of all previous attempts, with clues from unsuitable candidate solutions also being part of the deduction process. This is a type of ‘blind’ optimisation problem where no information is available on what makes a good solution, only information on how good solutions are.

Given a few initial guesses the player will *select* high scoring attempts and perform *crossover* to see if this results in an improvement. New colours will almost certainly have to be *mutated* into the ‘educated guesses’ in the attempt to find the correct sequence.

Fig 4-1 demonstrates how these three operators work considering a scoring scheme where a point is scored only for a number in the correct position.

The GA search procedure is very easy to understand and implement, with nature providing ready examples of exactly how things could be done.

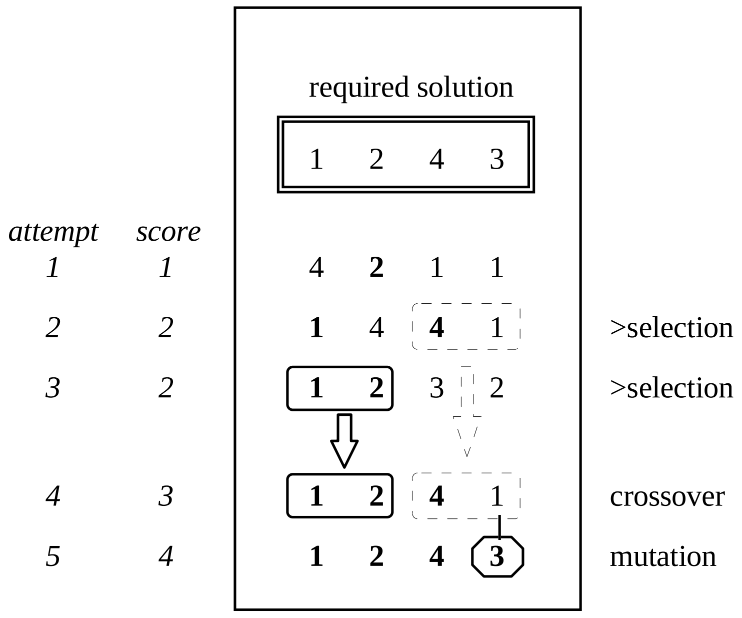

Fig 4-1 *An example of how the required solution evolves using the selection, crossover and mutation operators*

4.4 Implementation

4.4.1 Encoding

In optimisation problems a set of parameters is sought that will give the best solution to a particular problem. In order to implement a GA these parameters must be encoded into a string so that crossover and mutation can be applied. Binary encodings are the most common, due to the fact that Holland used them in his early pioneering work [**66**]. In DNA base 4 encoding is used, as the building blocks of DNA can take on 4 values, translated as A, C, G, or T.

Any base can be used, as it is just a different method of encoding the same information, but the lower the base the longer the string will be. For example, if a number is sought between 0 and 255 then this can be encoded as a binary string of length 8, a base 4 string of length 4, a base 16 string of length 2 or a base 256 string of length 1, as shown in Table 4-1.

Table 4-1 *Representations of the base 10 numbers 0 and 255 in different bases*

base 2 length=8	base 4 length=4	base 10 length=3	base 16 length=2	base 256 length=1
00000000	0000	000	00	0
11111111	3333	255	FF or |15|15|	|255|

It is clear that the importance of the operators will change depending on the base used. In base 2, the two given strings contain all the information required to derive any number between 0-255 by crossover alone. In base 16, two strings can at most lead to only 4 different numbers by crossover alone, mutation being required to introduce new information. In base 256 crossover cannot occur, mutation being the only operator that can introduce new numbers and finding a specific number becomes a pure random search.

In choosing an encoding scheme the nature of the problem will play a major role. If many real valued numbers are required in a solution then binary encoding becomes impractical as the string length increases. In [**67**] a method of selective genome growth is proposed that helps solve the problem of choosing how to represent a genetic algorithm.

4.4.2 Population Size

The population size is the number of candidate solutions in any one generation. In natural evolution the total population size is governed by what is sustainable by the environment and similarly in GAs the larger the population size the more computationally intensive (in terms of memory requirement) is the search.

In nature, the bigger the gene pool the more diverse is the genetic make up of the population with many individuals each with their own set of characteristics that enable them to survive. One advantage of this diversity is that there will be no dominant gene that, for instance, may be susceptible to a particular disease and result in the elimination of the whole species. In the bird family, sub-species have evolved with dominant characteristics that allow them to survive their local conditions and in effect are sub-optimal solutions in the search for a global ‘super-bird’. With large populations it can be seen how the search for the global optimal solution can be a slow (if not never-ending) process.

If the population size is small (e.g. a pride of lions), then a strong individual quickly becomes dominant and the diversity of the gene pool is restricted. The result is that good individuals (local optima) are quickly created but the dominance of particular genes restricts the search space. The chance of evolving the ultimate ‘super-lion(ess)’ (global optimum) is severely limited and would depend on mutation introducing new genes to diversify the search.

As new solutions are generated it is common to keep the population size constant by eliminating individuals (or letting them die), although this does not have to be the case. Ideas for the selection procedure for elimination are plentiful in nature. For example, each generation could be completely replaced by its offspring, or as a new offspring is created it could be accepted or rejected depending on its fitness. The advantage computers have over nature is that good individuals do not have to die and can be retained for indefinite reproduction. The retention of certain fit individuals is known as ‘elitism’.

4.4.3 Selection

This is the procedure for choosing individuals (parents) on which to perform crossover in order to create new solutions. The idea is that the ‘fitter’ individuals are more prominent in the selection process, with the hope that the offspring they create will be even fitter still.

Two commonly used procedures are ‘roulette wheel’ and ‘tournament’ selection. In roulette wheel, each individual is assigned a slice of a wheel, the size of the slice being proportional to the fitness of the individual. The wheel is then spun and the individual opposite the marker becomes one of the parents. In tournament selection several individuals are chosen at random and the fittest becomes one of the parents.

4.4.4 Crossover

Along with mutation, crossover is the operator that creates new candidate solutions. A position is randomly chosen on the string and the two parents are ‘crossed over’ at this point to create two new solutions. Multiple point crossover is where this occurs at several points along the string. A crossover probability (P_c) is often given which enables a chance that the parents descend into the next generation unchanged.

4.4.5 Mutation

After crossover, each bit of the string has the potential to mutate, based on a mutation probability (P_m). In binary encoding mutation involves the flipping of a bit from 0 to 1 or vice versa.

4.5 Experiments with GAs

4.5.1 Chinese Hat Optimisation Problem

To empirically evaluate the importance of the various parameters and techniques in GAs, several optimisation tests were performed. The code used is based on that in Appendix D. The experiments used tournament selection and a constant population size with the offspring replacing the parents every generation.

The fitness evaluation function (fitness landscape, scoring template) of candidate solutions for the first optimisation problem examined is shown in Table 4-2. For reference purposes this problem has been named the Chinese Hat because the scoring template diverges linearly outwards from the centre. There are two possible solutions for maximum fitness, one of which is shown by candidate solution 2, the other is the inverse of this where all the bits flip. The total number of candidate solutions is $2^{(\text{string length})}$.

Table 4-2 *An example of how the fitness of the solutions to the Chinese Hat problem are evaluated for a string length of 8. Each bit value in a solution is multiplied by the value in the same position in the scoring template and the total fitness is the square of the sum of all the bit scores. Each bit can have a value of 1 or –1*

Scoring Template	**4**	**3**	**2**	**1**	**-1**	**-2**	**-3**	**-4**

Candidate Solution 1	*1*	*1*	*-1*	*1*	*-1*	*-1*	*1*	*-1*
Bit by bit Score 1	4	3	-2	1	1	2	-3	4
Total Score 1 = 10^2 = 100								

Candidate Solution 2	*-1*	*-1*	*-1*	*-1*	*1*	*1*	*1*	*1*
Bit by bit Score 2	-4	-3	-2	-1	-1	-2	-3	-4
Total Score 2 = $(-20)^2$ = 400								

In the experiments, tests for each particular parameter setting were repeated to convergence 200 times to determine the average number of generations required to find the solution. Each subsequent trial differed by randomly generating a new initial population. After each crossover, mutation was only allowed on one randomly selected bit and whether it occurred depended on P_m.

4.5.2 Results

The results of varying the GA parameters for the Chinese Hat optimisation problem are shown in Fig 4-2 to Fig 4-8. All comments and discussion related to each figure are included below that figure.

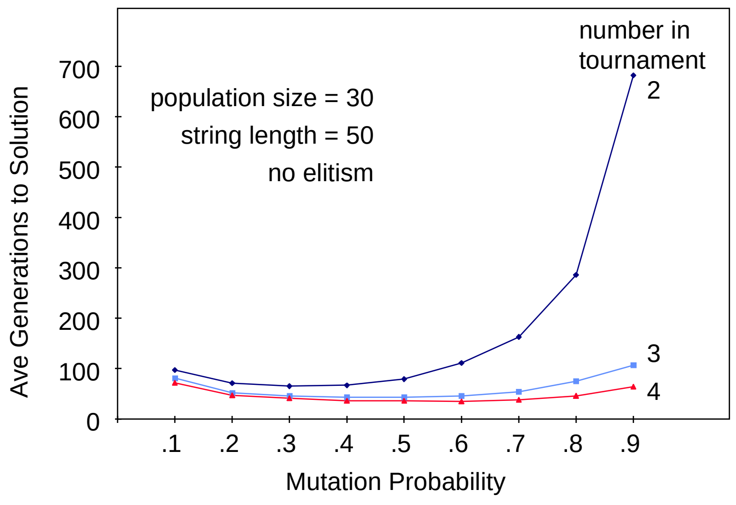

Fig 4-2 *The effects of* P_m *and the selection procedure*

By increasing the number of candidates (competitors) in the tournament for parenthood the number of generations required to convergence reduces. This would indicate that little diversity in the gene pool is required for this particular problem. There is also an optimum P_m around 0.5. With a higher mutation probability the number of generations starts to increase, although this becomes less significant as the selection procedure is made more competitive. In Fig 4-2, P_c =1.

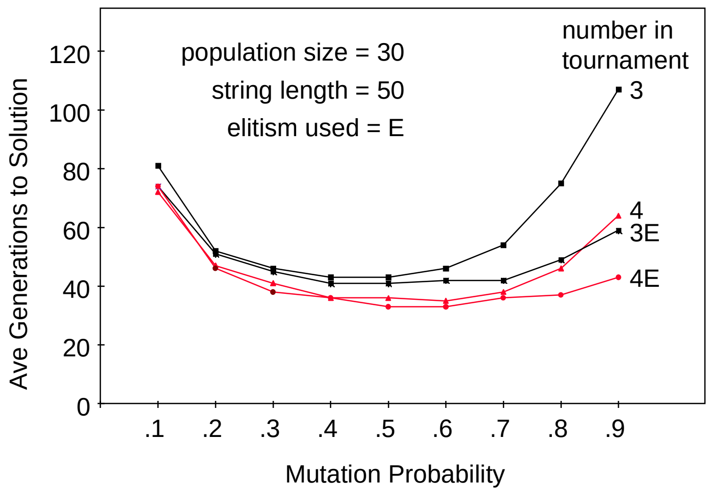

Fig 4-3 *Elitism*

The introduction of an elitist strategy, where the best individual is always retained, shows significant improvements in performance but only for the higher mutation rates, indicating the solution is evolved from mutation of this ‘best’ individual.

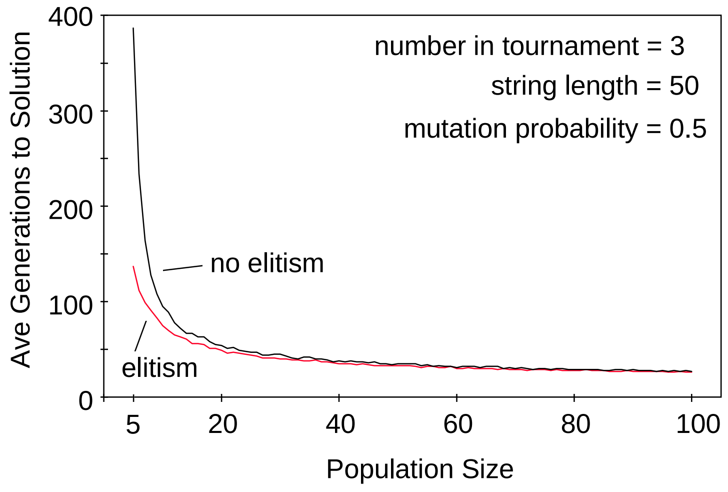

Fig 4-4 *Population size*

As would be expected, the larger the population size the fewer generations are required as the search space is increased. The highest rates of gain are seen by increasing the population size to 20, but even after this consistent reduction still occurs, as shown in Fig 4-5.

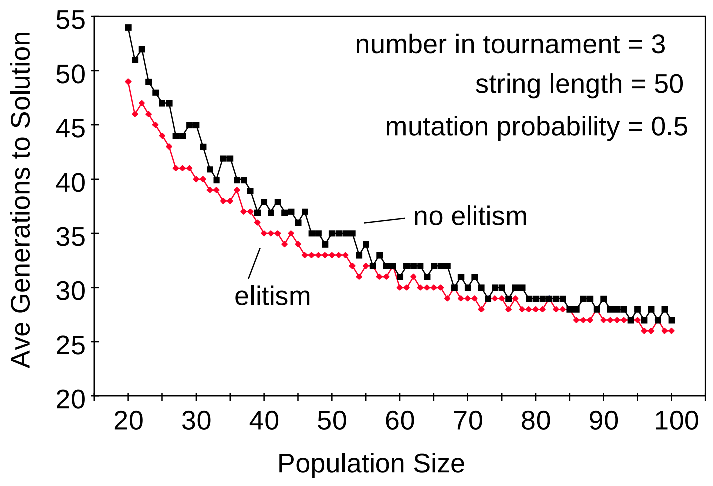

Fig 4-5 *Population size*

A close up of Fig 4-4 shows consistent improvement in performance with increasing population size.

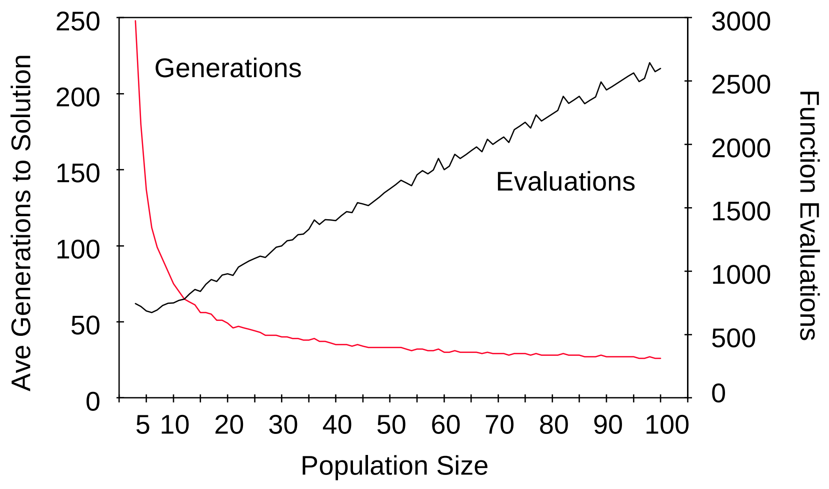

Fig 4-6 *Function evaluations*

In serial computing it is not the number of generations that is important but the number of function evaluations. That is, how many solutions must be evaluated before the optimum is reached, or roughly the number of generations multiplied by the population size. This gives an indication of the computing power (or time) required to solve the problem, assuming that evaluating the cost of each solution is a significant portion of the whole process. Fig 4-6 is the same data as that of the elitist tests in Fig 4-4, but with the number of evaluations also shown. It can be seen that for the given parameters, a population size of 6 is the most economical. After this the number of evaluations increases linearly with population size.

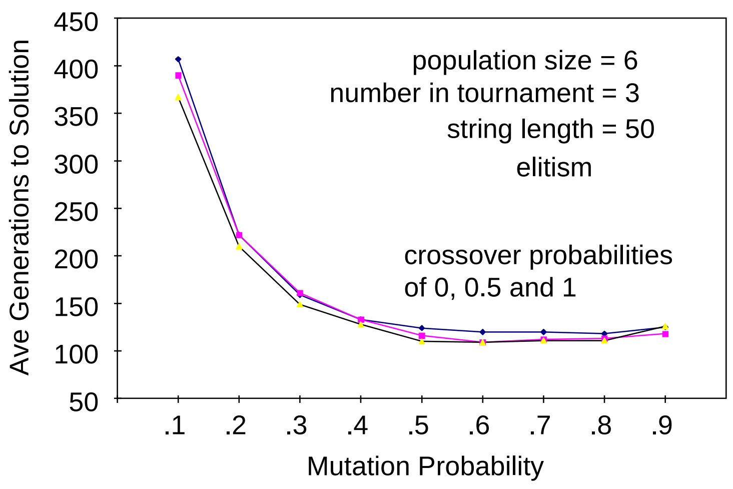

Fig 4-7 *The effect of crossover and mutation probabilities for a population size of 6*

Thus far crossover has occurred in every reproduction. By introducing a crossover probability, the relative importance of crossover and mutation can be examined. This is shown for the most efficient population size of 6 and crossover probabilities of 0, 0.5 and 1. What is seen is that the optimisation procedure for this small population relies solely on mutation with the crossover probability having a negligible effect.

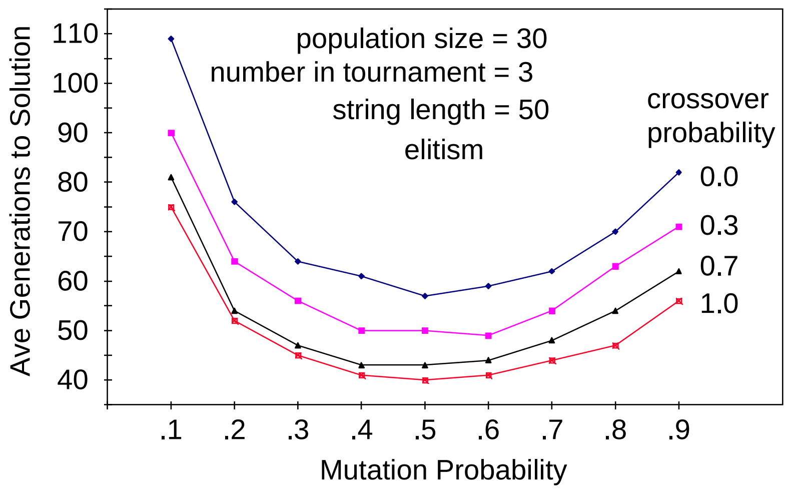

Fig 4-8 *The effect of crossover and mutation probabilities for a population size of 30*

With a larger population size increasing the crossover probability does improve the performance. Fig 4-8 is generated by the same procedure as Fig 4-7 but with a population size of 30. It can be seen with this larger population size there is an optimal P_m but improvements are also made by increasing P_c. What Fig 4-7 and Fig 4-8 show is not unexpected and is reflected in nature in that small populations rely on mutation for diversity whereas in larger populations it is a combination of crossover and mutation.

The experiments on this simple optimisation problem have illustrated that selecting the correct parameters is very important in genetic algorithms. What is also very evident is that there are definite relationships between all the parameters showing that fine-tuning is required to increase the speed to success, or reduce the chance of failure. The technique always managed to solve the problem, but how does it compare with other hill-climbing methods?

4.5.3 Other Iterated Hill-Climbing Methods

Other optimisation methods exist of which three were used for comparison with the GA. The following descriptions of these techniques are reproduced from [**68**].

4.5.3.1 Steepest-Ascent Hill Climbing (SAHC)

1. Choose a string at random. Call this string *current-hilltop.*

2. Going from left to right, systematically flip each bit in the string, recording the fitness of the resulting strings.

3. If any of the resulting strings give a fitness increase, then set *current-hilltop* to the resulting string giving the highest fitness increase (ties are decided at random).

4. If there is no fitness increase, then save *current-hilltop* and goto step 1. Otherwise goto step 2 with the new *current-hilltop.*

5. When a set number of function evaluations have been performed (here, each bit flip in step 2 is followed by a function evaluation), return the highest hilltop that was found.

4.5.3.2 Next-Ascent Hill Climbing (NAHC)

1. Choose a string at random. Call this string *current-hilltop*.

2. For *i* from 1 to *l* (where *l* is the length of the string), flip bit *i*; if this results in a fitness increase, keep the new string, otherwise flip bit *i* back. As soon as a fitness increase is found, set *current-hilltop* to that increased fitness string without evaluating any more bit flips of the original string. Go to step 2 with the new *current-hilltop*, but continue mutating the new string starting immediately after the bit position at which the previous fitness increase was found.

3. If no increase in fitness were found, save the *current-hilltop* and goto step 1.

4. When a set number of function evaluations has been performed, return the highest hilltop that was found.

4.5.3.3 Random-Mutation Hill Climbing (RMHC)

1. Choose a string at random. Call this string *current-hilltop*.

2. Choose a bit at random to flip. If the flip leads to an equal or higher fitness, then set *current-hilltop* to the resulting string.

3. Goto step 2 until an optimum string has been found or until a maximum number of evaluations has been performed.

4. Return the current value of *current-hilltop.*

1000 trials of each of these three algorithms were performed on the Chinese Hat problem for a string length of 50. The average number of evaluations, given in Table 4-3, shows that a GA is not the best method of solving this particular problem. In fact NAHC always reaches a global solution by traversing the string just once because the Chinese Hat is a smooth function when traversed from left to right.

Table 4-3 *The average number of function evaluations over 1000 trials for the Chinese Hat problem with a string length of 50*

SAHC	NAHC	RMHC	best GA	MRMHC
1082	48.6	190	650	430

The best GA based performance had a population size of 6 with a randomly selected gene being mutated with a probability of 0.5. With these parameters it was shown that crossover had a very limited effect (Fig 4-7, on page 84). As mutation appears to be the dominant process a variation of RMHC we called *multiple random mutation hill climbing* (MRMHC) was tried on the Chinese Hat function. This is basically RMHC but with each gene having a mutation probability as opposed to one randomly selected gene being mutated. Another way of describing MRMHC is a GA with a population size of 1 and the mutated offspring only surviving if it is as good as or better than the parent.

Fig 4-9 (on page 88) shows the average performance of MRMHC over 1000 tests with varying mutation probabilities and string lengths. The circled points represent the optimum mutation probabilities for the various string lengths with a pattern emerging that a P_m of (1/string length) appears to be the optimum. On average this is equivalent to 1 bit change per mutation which is an unsurprising result. Any less than this and some evaluations will be wasted as there will be no change, any more and there will be problems in fitting the last bit into position as there is a higher probability that more than one bit will be changed at once. The best performance with MRMHC for a string length of 50 was 430 evaluations, which occurred with P_m at just over 1/50 or 0.02.

The optimum mutation rate for MRMHC is on average 1 bit change per string, which is almost similar to RMHC, in which only one bit change per string is permitted. Similar results would be expected but it is noted that MRMHC requires over twice as many evaluations as RMHC. Why should this be so?

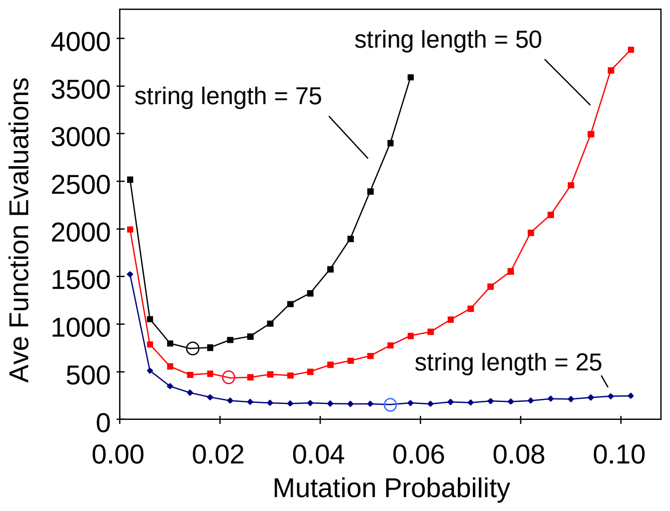

Fig 4-9 *The effect of mutation probability and string length for Multiple Random Mutation Hill Climbing optimisation*

If there is only one bit flip that is required to reach the optimum then RMHC should find this in an average number of evaluations equivalent to the string length. In MRMHC it is possible that in three consecutive evaluations the number of bit flips is 0,1 and 2, giving an average of 1. An evaluation with 0 flips is wasted and because only one bit requires correction, two bit flips will never find the optimum. It can thus be hypothesised why MRMHC gives a worse performance than RMHC for this particular problem.

4.5.4 Royal Road Functions

So what kind of problems will GAs be superior at solving than other search techniques?

The Schema Theorem and Building Block Hypothesis [**66, 69**] play on the idea that solutions are made up of short blocks of fit schema that use crossover to build up these schema into desirable solutions. A set of functions known as the 'Royal Roads' [**68, 70**, **71**, **72**] were developed that provide a fitness landscape designed specifically to be easily solvable by GAs if they did work in this building block manner. As described by the developers (Mitchell et al.), '*given the building block hypothesis, one might expect*

Table 4-4 *The Royal Road* (R$_1$) *fitness function. A bit string of length 64 contains 8 short schema that are the building blocks of the optimal schema. The wildcard '*' represents a 0 or 1 (or 'do not care'). The fitness of each candidate solution increases with the number of these building blocks present.*

11111111 ******** ******** ******** ******** ******** ******** ********	Schema 1	= 8
******** 11111111 ******** ******** ******** ******** ******** ********	Schema 2	= 8
******** ******** 11111111 ******** ******** ******** ******** ********	Schema 3	= 8
******** ******** ******** 11111111 ******** ******** ******** ********	Schema 4	= 8
******** ******** ******** ******** 11111111 ******** ******** ********	Schema 5	= 8
******** ******** ******** ******** ******** 11111111 ******** ********	Schema 6	= 8
******** ******** ******** ******** ******** ******** 11111111 ********	Schema 7	= 8
******** ******** ******** ******** ******** ******** ******** 11111111	Schema 7	= 8
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111	Schema Opt	= 64
11111111 11110011 01110011 ***11111111*** ***11111111*** 00000001 11110010 ***11111111***	e.g. Score	= 32

that the building block structure of R$_1$ will lay out a "royal road" for the GA to follow to the optimal string'. Table 4-4 shows one of these Royal Road functions, R$_1$.

In their analysis, Mitchell et al. used a GA with a population size of 128, single point crossover with P_c fixed at 0.7 and P_m at 0.005, full details are given in [**68**]. Over 200 runs the mean number of GA function evaluations was 61,334, an order of 10 times higher than RMHC (6,179). NAHC and SAHC never reached the optimum solution, which is not unexpected given the nature of the fitness landscape.

In section 4.5.2 the importance of the GA parameters was demonstrated, although only on a simple smooth function that proved easier to solve by other methods. The Royal Road problem was investigated in the same manner to determine if the nature of the problem affected the relationship between the parameters.

Initial tests were performed to see if the results of Mitchell et al. could be replicated and also to examine the effect of varying the GA parameters. Mutation probabilities between 0.002 (0.13 in 64) and 0.05 (3.2 in 64) were tested for crossover probabilities between 0 and 1 inclusive. Each set of parameters was repeated to convergence 20 times and the mean value recorded. Tournament selection was used where each parent was the best of 5 randomly chosen candidates. The results are shown in Fig 4-10.

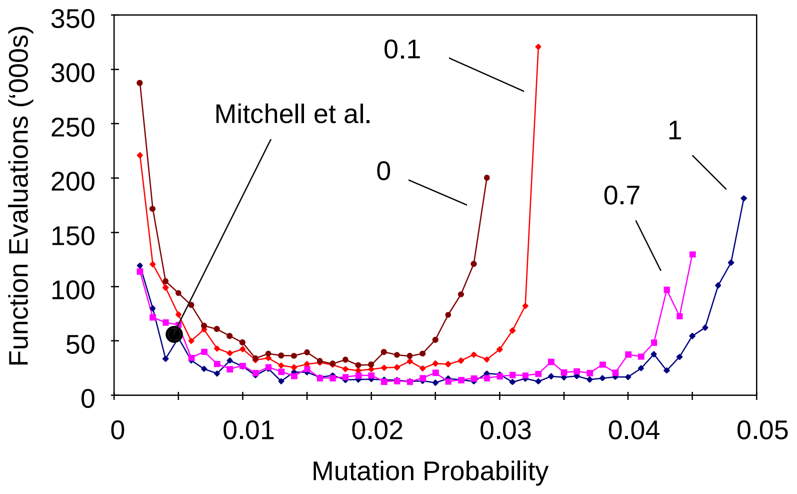

Fig 4-10 *The effect of the mutation probability for four crossover probabilities (0,0.1,0.7,1) on the Royal Roads (R_1) landscape. Each point is the average over 20 tests with a population size of 128. Mitchell et al. used a mutation probability of 0.005 (0.33 in 64) and crossover probability of 0.7 that gave a mean of 61,334 function evaluations to convergence over 200 tests.*

The results of Mitchell et al. were easily replicated even though a different selection procedure was used. By increasing P_m to 0.02 (1.3 in 64) the number of evaluations was reduced to around 14,000, a factor of 4 improvement and only twice as many as RMHC. With this mutation probability the function could be optimised in 28,000 evaluations without using crossover at all, half as many as the evidently poorly tuned GA of Mitchell et al. With no crossover, each offspring is a mutation of a parent chosen due to its fitness.

It has been demonstrated that a GA with no crossover can outperform a poorly tuned GA on a fitness landscape purposely designed to suit the crossover operator. If it can be discovered what determines a good mutation probability with no crossover then this should be generally applicable when crossover is applied.

With no crossover, the relationship between the mutation probability and selection procedure was examined. In previous tests on the R_1 landscape tournament selection was used where each parent was the fittest of 5 randomly selected candidates. The number of candidates was varied along with P_m, as shown in Fig 4-11 (on page 91). What is clear is that the less stringent the selection, the tighter the band is for an acceptable value of P_m.

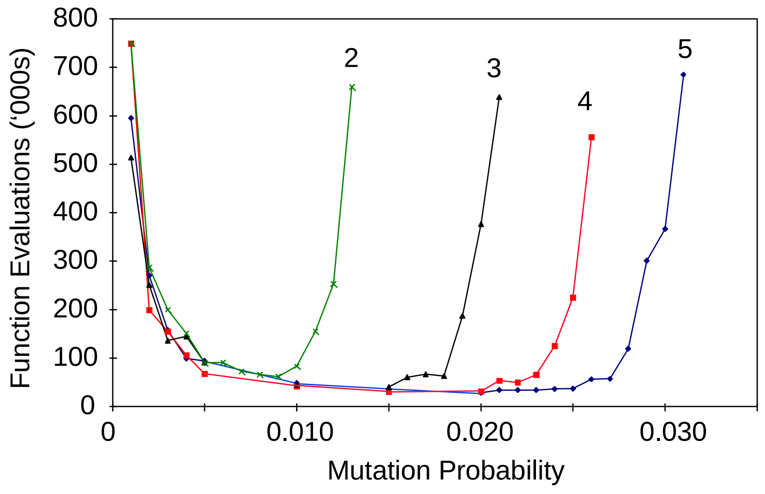

Fig 4-11 *The relationship between the mutation probability and the number of contestants in tournament selection (2,3,4, and 5). The crossover probability is 0 and the population size is 128. The mean of 20 trials was recorded.*

For each selection policy there is also an upper mutation probability past which the required evaluations increase exponentially; the more stringent the selection then the higher is this upper limit.

The objective of this exercise was to discover what determines a good mutation probability, which has been shown for R_1 to also depend on the selection procedure used, becoming more important the weaker the selection procedure. There is a definite lower limit around 0.005 or 0.33 in 64.

In order to determine a desirable mutation rate the effect of the population size must also be investigated. Fig 4-12 (on page 92) shows that given a near optimal P_m (0.01) there is little sensitivity to population size, but as P_m increases so does the sensitivity to population size.

The conclusion reached thus far is that the mutation probability is the most important GA parameter in solving the R_1 landscape. There is also much evidence (and common sense) to suggest that the optimum mutation probability is related in some way to the string length. In order to test this theory MRMHC (a GA with no crossover and population size 1 with the new solution being retained if it is better than or equal to the parent)

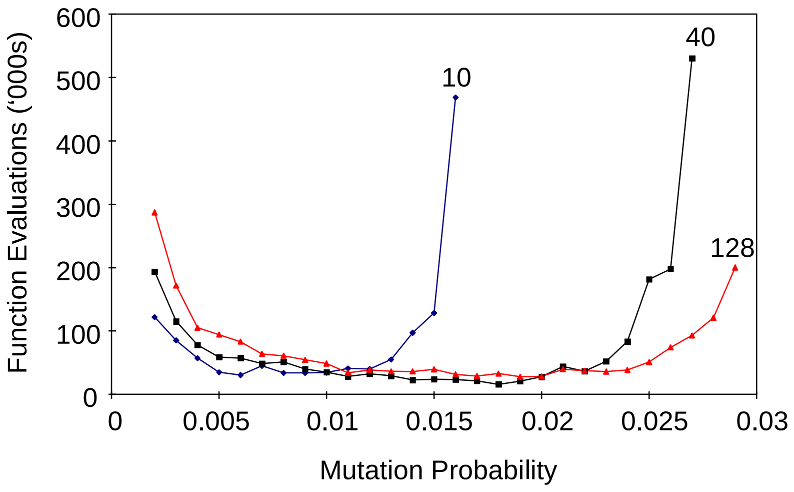

Fig 4-12 *The effect of the mutation probability and the population size (10,40,128). In these cases the P_c = 0 and the number of candidate parents is 5.*

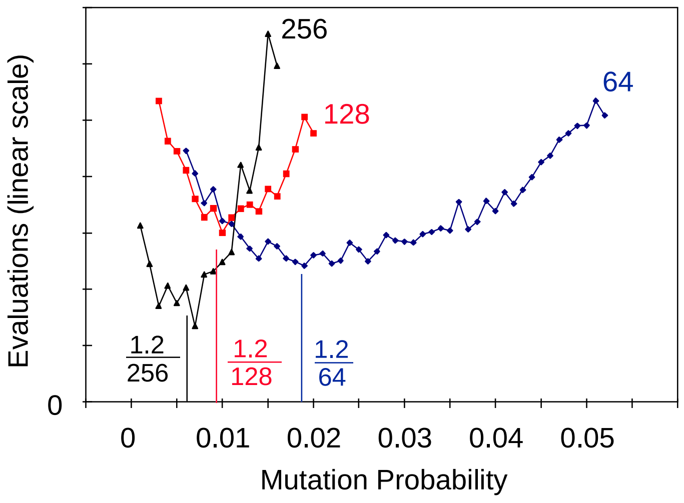

Fig 4-13 *The effect of the mutation probability on MRMHC, averaged over 200 tests. The optimum is (1.2/64)***.**

was used to solve three versions of R_1, with string lengths of 64, 128 and 256. The results in Fig 4-13 show that the longer the string the more sensitive is the search to P_m (the y-axis scale in Fig 4-13 is different for each population size). The optimum value of P_m was observed to be about (1.2/string length). The question to be investigated now is what is special about this mutation rate?

In their work, Mitchell et al. analysed the RMHC algorithm with a simple derivation based on probability that gave the expected number of function evaluations to solve R_1.

Consider R_1 as in Table 4-4. In each schema of length 8 the number of possible combinations is 2^8. If one and only one bit is changed in each evaluation then the chance of this bit being in a specific schema is 1/8, since there are 8 schemas in total. Thus the chance of randomly creating a particular schema is once every $2^8 \times 8$ evaluations. Initially there are eight schemas to choose from so the chance of creating any schema is once in every $2^8 \times 8/8$ evaluations. Once one schema is found the chance of finding a further schema decreases to 7/8 of that of finding the first since 1 in 8 bit changes are likely to be wasted changing the already discovered schema. The number of evaluations required to find this second schema thus increases to 8/7 that required to find the first. The expected number of evaluations to find a single schema is in fact slightly more than 2^8 and as determined by a Markov-chain analysis it is 301.2 [**68**]. The expected number of evaluations to solve the problem is thus,

$$301.2\times 8\times\left(\underset{\substack{\uparrow\\ \text{1st}}}{\frac{1}{8}}+\frac{1}{7}+\frac{1}{6}+\frac{1}{5}+\frac{1}{4}+\frac{1}{3}+\frac{1}{2}+\underset{\substack{\uparrow\\ \text{8th}}}{\frac{1}{1}}\right)$$

discovered schema

Tests were performed for RMHC that tracked the creation of the schema in the solution in order to confirm the theoretical performance. Table 4-5 shows the results averaged over 1000 trials which almost mirror the theoretical expectations.

Table 4-5 *The theoretical and experimental (averaged over 1000 trials) number of evaluations to discover each subsequent schema for R_1 using RMHC. The total theoretical evaluations = 6,549, experimental = 6,542 and Mitchell et al. = 6,179.*

schema	1	2	3	4	5	6	7	8
theoretical evaluations	301.2	344.6	401.6	481.9	602.4	803.2	1204.8	2409.6
experimental evaluations	284	355	384	508	622	797	1182	2410

RMHC has been shown to behave as the probability theory predicted. In GAs the theory of how they behave remains a theory, with little experimental evidence to try to observe their actual behaviour.

Tests were performed using GAs on the R_1 landscape that tracked the formation of the schema. The trials were performed 500 times with the maximum number of generations set at 800. The maximum, minimum and average fitness of the population were recorded at each generation and averaged for the trials that had not converged. Initially the crossover rate was set at 0.7, population size 128 and the number of competitors in the tournament was 5. Three mutation rates were used, 0.33/64, 1.3/64 and 2.7/64. The results are shown in Fig 4-14 to Fig 4-17.

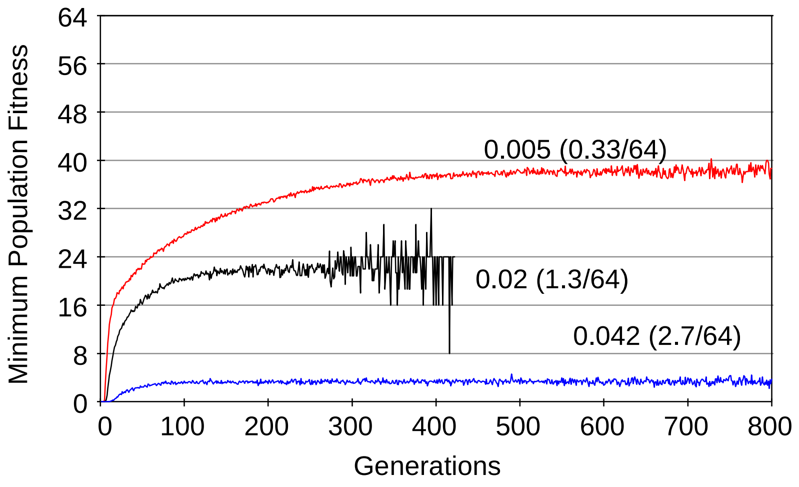

Fig 4-14 *The effect of the mutation rate on the minimum population fitness*

The lower the mutation rate the fitter is the worst individual in the population. Note that in all cases the minimum fitness reaches a plateau and only for the middle mutation rate of 1.3/64 do all the trials converge.

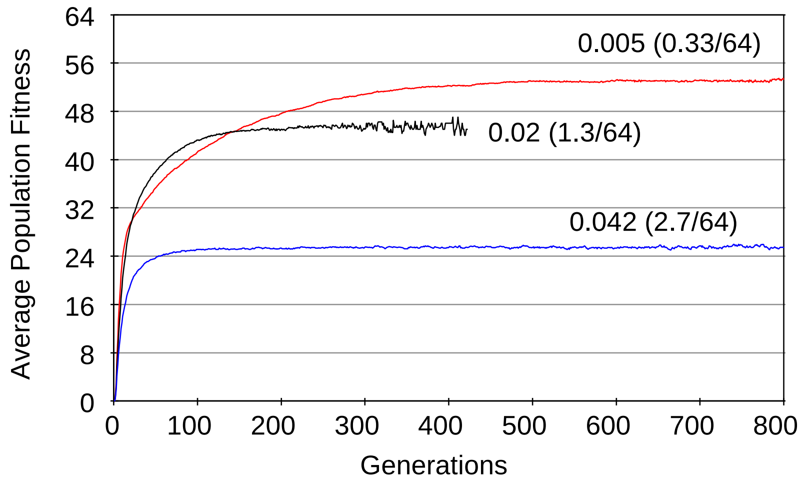

Fig 4-15 *The effect of the mutation rate on the average population fitness*

It can be seen how the rise in average population fitness is initially high for all cases. The best average population is with the lowest mutation rate, but this does not find the global solution in all cases.

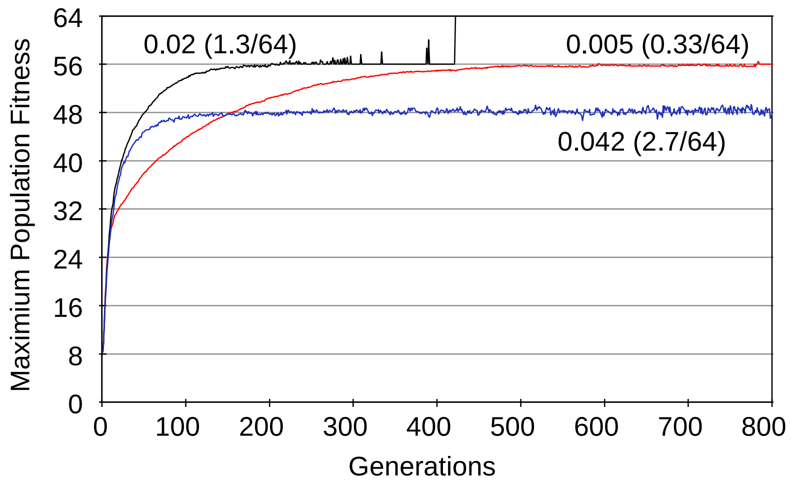

Fig 4-16 *The effect of the mutation rate on the maximum population fitness*

Note that the high mutation rate generally limits the maximum population fitness to 6 schema (a fitness of 48). This is because schema are destroyed as new ones are created.

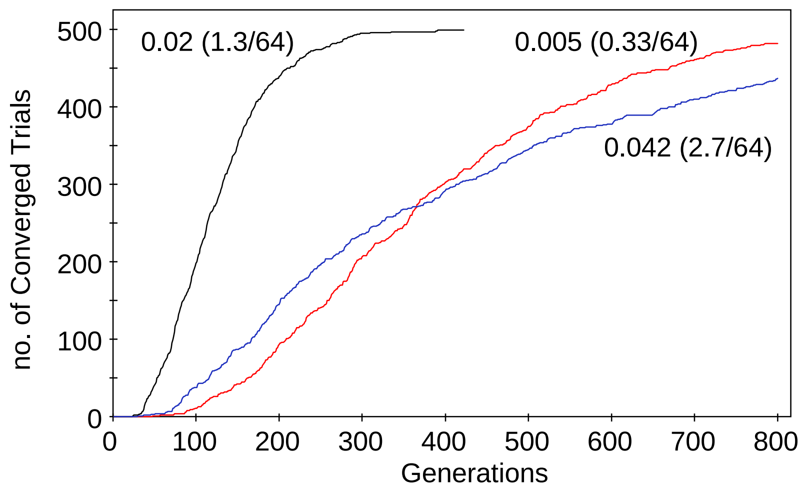

Fig 4-17 *The number of converged trials at each generation for the three mutation rates*

Quite clearly from a pure optimisation perspective, where the goal is to find the global solution, the mutation rate of 1.3/64 is superior in all respects, as shown by Fig 4-17.

Fig 4-18 to Fig 4-21 show the effect of the crossover probability for the near optimum mutation rate of 1/64. It can be seen that increasing P_c only improves the speed to convergence, with no other effect on the behaviour of the GA, as identified by all the lines converging to the same fitness value. In all cases every trial converged, even with P_c=0. The conclusion drawn is that mutation is the most important operator for this particular problem.

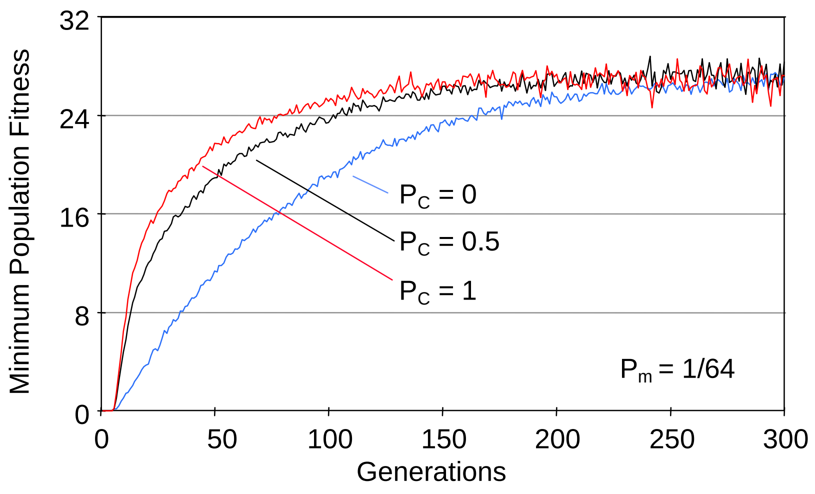

Fig 4-18 *The effect of P_c on the minimum fitness*

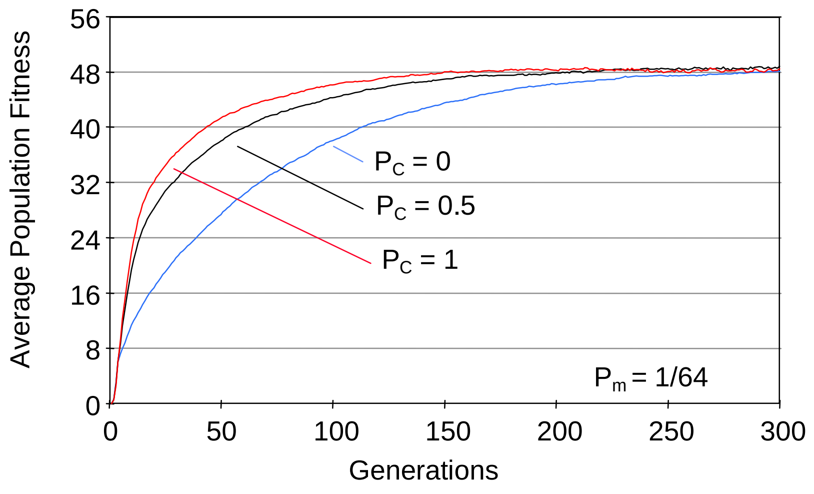

Fig 4-19 *The effect of P_c on the average fitness*

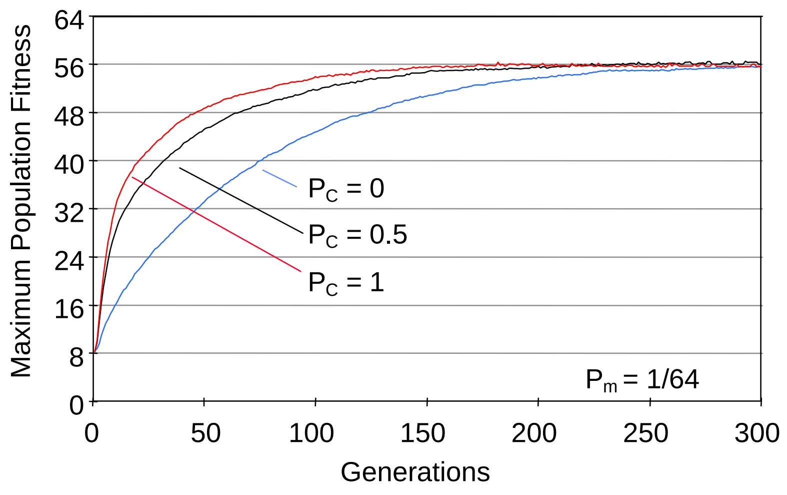

Fig 4-20 *The effect of P_c on the maximum fitness*

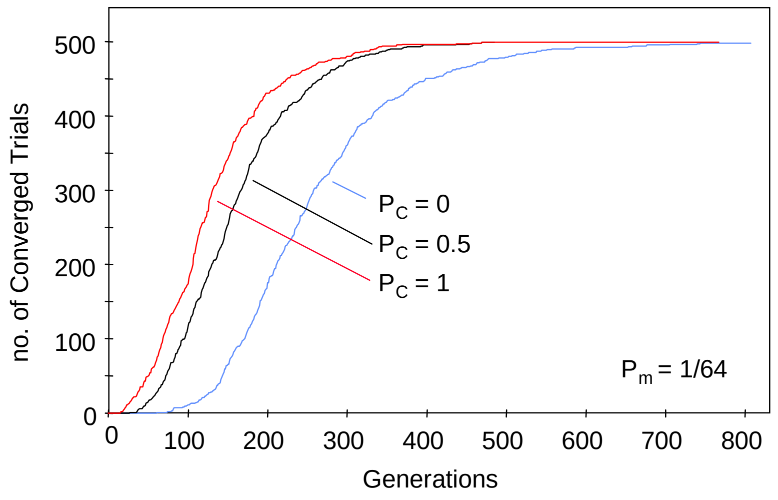

Fig 4-21 *The number of converged trials at each generation for the three crossover probabilities*

4.6 Chapter Summary

The work in this chapter has been an empirical investigation of parameters that affect GA performance. As commented in [**73**], '*there is a growing realisation that results obtained empirically are no less valuable than theoretical results*'.

What has been concluded is summed up in [**74**], '*From a function optimization point of view, GAs frequently don't exhibit a killer "instinct" in the sense that, although they rapidly locate the region in which a global optimum exists, they don't locate the optimum with similar speed*'.

This 'killer instinct' has been shown to be dependent on the mutation rate, which is critical for efficient GA performance in global optimisation. In humans, characteristics of individuals that enable them to stand out from the norm are often a result of mutation. This is exemplified by Veikko Hakulinen, a Finnish cross-country skier who won medals in the 50k, 30k, 15k and 4x10k relay at the 1956 winter Olympics. On medical examination it was found that he had an excessive red blood cell count that enabled him to take in more oxygen and not become out of breath. This was caused by a genetic defect with a probability of occurring equal to that of picking a specific light bulb with all the light bulbs on earth to choose from.

There has been much theoretical academic work in trying to improve the efficiency of GAs by optimising parameter settings. In other work, previously claimed ‘good’ settings are taken and used on totally unrelated problems. If GAs are to be used for function optimisation then a thorough investigation of the parameters is required.

It must be remembered that ‘*Genetic algorithms are NOT function optimizers*’ [**74**] and that other techniques do exist, that, although they do not sound as interesting, may be more appropriate for solving a particular class of problem. Optimising a system where there is no information on the dynamics (‘black box optimisation’) is essentially a directed random search, with the direction being guided by the strategy used. The purpose of these strategies is to guide the search to increase the probability that in time, a solution will be found. As was demonstrated (see Table 4-5), on average over many trials, random mutation hill climbing behaves exactly as a Markov chain analysis predicts. Nix and Vose [**75**] performed a similar Markov chain analysis for a simple genetic algorithm and claim that ‘*if the finite population is sufficiently large, we can accurately predict the convergence behaviour of a real GA*’.

Along with GA’s, simulated annealing [**76**] is another popular strategy for ‘black box’ optimisation that is inspired by nature. This is based around the fact that close temperature control must be maintained when cooling liquids into solids in order to attain a specific lattice structure. The most energy efficient lattice structure is obtained by very slow cooling and sometimes slight heating. This is reflected in the optimisation by only applying slight random perturbations and limiting the ‘temperature gradient’ (the amount of improvement allowed in new solutions). Successive solutions are also allowed to be ‘hotter’ (or worse) than previous attempts.

Many other optimisation strategies exist [**77**], including and tabu search [**78**] and branch and bound [**79**] (branch and bound methods are not strictly black box since they rely explicitly on the cost structure of partial solutions [**80**]).

In conclusion ‘*for any algorithm, any elevated performance over one class of problems is offset by performance over another class*’ [**80**].

In chapter 5 a variation of RMHC is used for the optimisation of a domestic hot water tank based on real-time pricing of electricity.

[66] J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[67] L. Altenberg, Evolving better representations through selective genome growth,' *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, vol.1, 1994, pp. 182-187

[68] M. Mitchell, *An introduction to Genetic Algorithms*, The MIT press, 1996

[69] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[70] M. Mitchell, S. Forrest, and J.H. Holland, "The royal road for genetic algorithms: Fitness landscapes and GA performance," *Proceedings of the First European Conference on Artificial Life*, 1992, pp. 245-254.

[71] S. Forrest and M. Mitchell, "Relative building-block fitness and the building-block hypothesis," In *Foundations of Genetic Algorithms 2*, Morgan Kaufmann, 1993, pp.109-126.

[72] M. Mitchell, J.H. Holland, and S. Forrest, "When will a genetic algorithm outperform hill climbing?," In *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, 1994.

[73] T. Walsh, "Empirical methods in AI," *AI Magazine*, summer 1998, pp.121-124.

[74] K.A. De Jong, "Genetic algorithms are NOT function optimizers," In *Foundations of Genetic Algorithms 2*, Morgan Kaufmann, 1993, pp.5-17.

[75] A.E. Nix and M.D. Vose, "Modeling genetic algorithms with Markov chains," *Annals of Mathematics and Artificial Intelligence*, vol. 5, 1992, pp.79-88.

[76] S. Kirkpatrick, D.C. Gellat and M.P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, 1983, pp. 671-680.

[77] A.H.G. Rinnooy Kan and G.T. Timmer, "Global optimisation: a survey," International Series of Numerical Mathematics, vol. 87, 1989, pp. 133-155. Bikhauser Verlag Basel.

[78] F. Glover, "Tabu search I," *ORSA J. Comp*., vol. 1, 1989, pp. 190-296.

[79] E.L. Lawler and D.E. Wood, "Branch and bound methods : a survey," *Journal of Oper. Res*., vol. 14, 1966, pp. 699-719.

[**80**] D.H. Wolpert and W.G. Macready, “No free lunch theorems for optimization,” *IEEE Trans. Evolutionary Computation*, vol. 1, no. 1, April 1997, pp. 67-82.