# A Study of Genetic Algorithms solving a Combinatorial Puzzle

Thomas Wolf
Queen Mary & Westfield College, University of London,
Mile End Road, London E1 4NS, UK
email: T.Wolf@maths.qmw.ac.uk

January 20, 1998

**Abstract**

The suitability of Genetic Algorithms (GAs) to solve a combinatorial problem with only one solution is investigated. The dependence of the performance is studied for GA-hard and GA-soft fitness functions, both with a range of different parameter values and different encodings.

## Contents

# List of Figures



# 1 Introduction

The application of Genetic Algorithms (GAs) to be discussed in this paper arose as an example in teaching GAs. The aim in this paper is to clarify the effect of a variation of GA-parameters and of the evaluation function by doing a statistical analysis of a large numbers of computations. What is also demonstrated in this study is that GAs can successfully be applied to the solution of combinatorial problems with a unique solution in contrast to conventional applications of GAs where an as good as possible approximative solution is looked for in a limited time. The study of parameter dependence is supposed to help dealing with much harder combinatorial problems than what is solved below.

The other suggestion is a simple measure to partially compensate loss of genetic diversity by increasing the ratio of mutation/crossover.

The puzzle to be solved by the GA is to map each letter in the following figure to a digit such that the 3 horizontal and 3 vertical computations are correct.

```
EDKH  /  KF  =    AA
  -      +        +
 EDB  x   J  =  EHCG
------------------
EEJD  -  DK  =  EEAE
```

With 10 letters and digits there are 10!=3628800 permutations from which only one `(ABC..K=5793146082)` solves the puzzle. This size of configuration space is large

enough to study non-trivial effects by comparing a GA-hard evaluation function with many local optima with a function with fewer local optima and a function with only one optimum. On the other hand this problem is small enough to solve it very often to visualize parameter dependence and not only to produce single numbers for average and variance of efficiency. Finally, the fact that this puzzle has a single solution provides a unique stopping criterion.

# 2 Evaluation functions

Three versions of evaluation functions are tested which are denoted 1 - 3 in the rest of the paper.

1. The naive and simplest way to measure progress is to count how many of the three horizontal and three vertical calculations are correct, i.e. the evaluation function returns a value in $0 \ldots 6$. Although the other two functions return more information and give a better feed back, this functions is used to study combinatorial problems for which no good evaluation function is known and which are therefore a bigger challenge for Genetic Algorithms, i.e. which are GA-hard. This function is even more 'GA-hard' than it looks. The reason is that only very close to the solution of the puzzle do chromosomes have a higher fitness than 2. So during nearly all of the evolution, chromosomes have only the three different fitness values 0,1,2.

2. An improved evaluation function compares digit by digit the result of a calculation with what is encoded in the puzzle. For example, in the first line after evaluating $(K10 + F)(A10 + A)$ the result is compared with $EDKH$ for each of the four digits. In this way 22 comparisons are made in all 6 calculations which gives a more detailed feedback. In the case of a subtraction the equivalent addition is checked and in the case of a division the corresponding multiplication is checked. This resulting evaluation function is refined further by a number of modifications:

   - A bonus of 10 is given if the rightmost position is correct whereas a correct comparison of other positions gives only a bonus of 8. The reason for these different weights is that a wrong carry over due to wrong digits to the right can make wrong digits look fine if their sum is off by only $\pm 1$.
   - For the same reason (correct digits looking wrong due to a false carry over), a bonus of 4 is given if a not-rightmost comparison gives a discrepancy of only $\pm 1$.
   - If the rightmost comparison in a multiplication is wrong then no other digits in this calculation are compared because a wrong multiplication usually inflicts a false carry over which prevents checking of digits further left.

3. A third function includes perfect information and is used only for comparisons. It counts the number of correct digits in the chromosome by comparing it with the unique correct chromosome which solves the puzzle.

The first of these functions has very many local optima, the second function less and the third by definition only one. This allows one to investigate the effect of changing GA-parameters for these quite different functions.

# 3 Generation of new chromosomes

After determining raw fitness values with one of the above fitness functions these values are slightly modified to initialize a roulette wheel for selecting parents to generate offspring. We used a simple shift to improve the chance of better chromosomes. If $m$ is the minimal raw fitness then each fitness $f$ is changed to $f := f - m/2$.

Offspring chromosomes are created with three operators:

- *Crossover*: The usual order-based crossover operator as, described in [1], is slightly modified to enable one to take advantage of additional knowledge of potential building blocks.

  According to the conventional method ([1],[2]) two child chromosomes are generated at the same time. A random bit-string $B$ (containing 0's and 1's) with the length of a chromosome is generated. For each '0' in this string the 1st of the children inherits the allele from the 1st of the parents at this position in the chromosome, and for each '1' the 2nd child inherits the allele from the 2nd parent. The remaining alleles of the 1st child that have not got a value yet are filled with the missing digits in the same order as they occur in the 2nd parent and the so far unassigned alleles of the 2nd child are filled with the missing digits as they occur in the 1st parent.

  The modification in our program is that the bit-string $B$ starts with a random first component $B_1$ (= 0 or 1) and each element $B_n$ of this string is determined from $B_{n-1}$ by

$$B_n = \begin{cases} B_{n-1} & \text{with probability } 1-p \\ 1-B_{n-1} & \text{with probability } p \end{cases} \tag{1}$$

  The advantage of this assignment of $B$ is that one has an extra parameter $p$ which is a probability of switching inheritance between both parents, i.e. it is an inverse measure of how lumpy are the parts that are copied from the parents. If one knows about the strong causal interdependence of some of the variables to be determined then one can encode the chromosome such that the corresponding genes are located next to each other in the chromosome. By having a smaller value of $p$ one has a higher probability that more than one allele in a row is copied from the parents to the children which increases the chance to

inherit pre-optimized building blocks of parent chromosomes. This is discussed in section 6.4.

If two parent chromosomes differ only in two positions, then crossover will generate two offsprings that are equal to their parents. In order to keep the genetic variety high, offsprings must not be a duplicate to any already existing chromosome. In the case of a clash, crossover is tried again and if it happens consecutively $c_r$ times then the mutation operator is tried instead in order to increase genetic variety. $c_r$ is called below the maximal-crossover-repeats parameter.

- *Small mutation*: Two positions in the chromosome are determined randomly and their values are exchanged.
- *Large mutation*: Two positions in the chromosome are determined randomly which determines an interval. With a chance of 50% all positions within this interval are permuted otherwise all positions outside this interval are permuted randomly.

Operator weights that determine how often each of these operators is used are kept constant. One reason for this is that running times may vary greatly with different parameter sets. Therefore a dynamic variation would be necessary, compensating dynamically for the loss of diversity during evolution. The maximal-crossover-repeats parameter $c_r$ plays such a role although only to a very limited extend. Its advantage is that its use comes for free, whereas monitoring genetic diversity would otherwise be more expensive. A more radical measure is to restart evolution, which is studied in section 6.7.

# 4 Selection to drop

The list of parents and the list of offsprings are appended and closed to form a circular linked list, with a pointer pointing to an arbitrarily defined start of this list. To delete a chromosome a random (integer) number $r$ in the interval

$$m \dots m + [s(M - m) + 0.5] \tag{2}$$

is generated where $m, M$ are the minimum and maximum of the fitness values of parents and offspring determined by the evaluation function and $s$ is a selectivity parameter giving an increasingly aggressive selection the smaller $s$ is. $[x]$ denotes the biggest integer $\leq x$. Then the pointer is advanced in the circular linked list until a chromosome is reached with a fitness $\leq r$ which is subsequently killed unless it has maximal fitness $M$.

In the process of dropping chromosomes, the minimum fitness of the remaining ones may increase if all chromosomes with minimal fitness $m$ are dropped. Therefore $m$ has to be updated when stepping through the circular linked list.

The aim was to have a parameter allowing different selectivities and also allowing chromosomes with a high fitness to be dropped. Compared with a roulette wheel selection we have no need to

1. total the fitness of all chromosomes,
2. assign to each chromosome its partial sum of fitness and
3. step through the list of chromosomes until their partial sum exceeds a random number.

The resulting speed up becomes increasingly useful for larger population sizes.

# 5 Alternative encodings

On first sight it does not seem to matter in which sequence the letters $A \dots K$ are encoded in the chromosome.

The essence of Genetic Algorithms - the combination of optimized building blocks of two parent chromosomes through the crossover, is more effective if such building blocks are passed as a whole to offspring and less likely to be cut through during crossover.

What are such building blocks in our problem? Looking, for example, on the right vertical calculation and the rightmost digits, it appears that if $A + G = E$ or $A + G = 10 + E$ is satisfied and one of the three digits $A, G, E$ is changed then at least another one of them has to change as well in order to keep the sum correct. Such a causal dependence is obviously not the case for each triple of letters, like $A, D, J$. Therefore $A, G, E$ should be grouped next to each other in the chromosome as a potential building block. Applying this principle consequently, grouping rightmost letters of each three numbers involved in any calculation next to each other we obtain, for example, the encoding HFAGEKDBJC. In section 6.4 the effect of different encodings is discussed for different evaluation functions.

# 6 Results

## 6.1 Specification of diagrams

Diagrams shown in this section display the number of chromosomes that had to be generated in order to solve the puzzle. On the horizontal axis the $\log_{10}$(number of chromosomes generated to solve the puzzle) is given and the vertical axis measures the frequency with which that number of chromosomes was necessary. For each graph the puzzle was solved between 3000 and 40000 times depending on whether the average number of generated chromosomes was high ($\sim 10^5$) or low ($\sim 2000$). The interval between the minimal and maximal number of chromosomes for each graph was divided

into 20 subintervals for 3000 runs and into 50 subintervals for 40000 runs. The number of runs with chromosomes in this subinterval was counted and constitutes the vertical measure after having been normalized so that the area under each graph in a figure is the same.

Each diagram displays the variation of few parameters, usually only one. The values of the other parameters which are held constant are given, using the following abbreviations:

ef ... evaluation function used (= 1,2,3 see section 2)
cd ... encoding used (= 1 for HFAGEKDBJC, 2 for ABCDEFGHJK)
s ... selectivity parameter ($s$ in (2))
nc ... number of children chromosomes generated in one go
np ... number of chromosomes in the population
p ... probability in % that during crossover an allele is inherited from a parent if the previous allele in the chromosome was inherited from the other parent ($p$ in (1))
cr ... maximal number of repeats of crossover if duplicate chromosomes are generated (see the description of crossover in sec. 3)
co ... probability in % of children generated by crossover
m1 ... probability in % of children generated by swapping two alleles
m2 ... probability in % of children generated by interval mutation
rf ... restart factor - a measure of how long no improvement must have happend in order to restart evolution, plays only a role in section 6.7

For all-or-nothing problems like our puzzle we are interested in the effort required to find the unique solution, and therefore we have not monitored the progress in time, i.e. its dependence on the number of chromosomes generated so far.

One could have plotted the integral of the graphs of our diagrams which would show the accumulated probability of solving the puzzle. By showing the probability density instead, a higher resolution of the right flank of our graphs is achieved which simplifies comparisons of graphs for high chromosome numbers.

Also, we are less interested in the actual time in which the puzzle is solved (about 1 sec on a 133MHz PC for evaluation function 2) but in relative differences when comparing different parameter values.

At first single parameters are varied; later combinations of parameters.

## 6.2 The size of population and number of children generated

In figure 1 below, the dependence on the number of children generated in one go is shown. As is to be expected, the influence of this parameter is small. The benefit of generating a smaller number in each generation (and consequently mor generations) is that if they amoung this generation there are any good individuals then they can have descendants already in the next generation.

In contrast, the size of the population is a much more critical parameter as shown in figures 2 and 3.

The situation is qualitatively the same for evaluation functions 1 and 2. A larger population size gives a smaller variation whereas small population sizes carry the risk of getting stuck at a local optimum and then about 100 times as many chromosomes have to be generated to solve the puzzle.

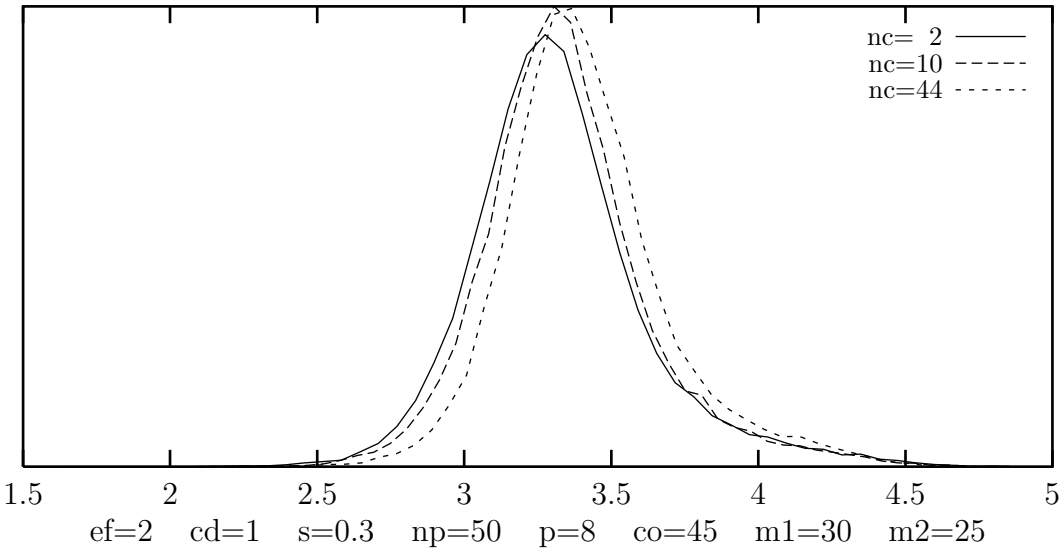


Figure 1: Dependency on the number of children

## 6.3 Variations in selectivity

If an evaluation function has many sub-optima then it is important that evolution does not concentrate around the first optimum it finds. This can be achieved with a selectivity that allows for good chromosomes to die as well. Therefore for evaluation function `ef=1`, variations of selectivity have a similar effect as different population sizes (figures 2, 4). If there is a chance that preliminary good chromosomes can be deleted, then the risk to get stuck with local optima is reduced. Consequently high values of $s$ are optimal. The shape of the curves changes relatively quickly for $s > 0.5$ but very little for $0.3 < s < 0.5$. The reason is that during nearly all of the evolution fitness values are only 0,1 and 2 and for $s < 0.5$ only chromosomes with fitness 0 are deleted. Evaluation function 2 in contrast has less local optima so a highly aggressive selection (low value of $s$) is optimal there (figure 5). The way selection is performed (only a few chromosomes are deleted for the next children to be generated) ensures that no second hump develops like in figure 3 for small population sizes.

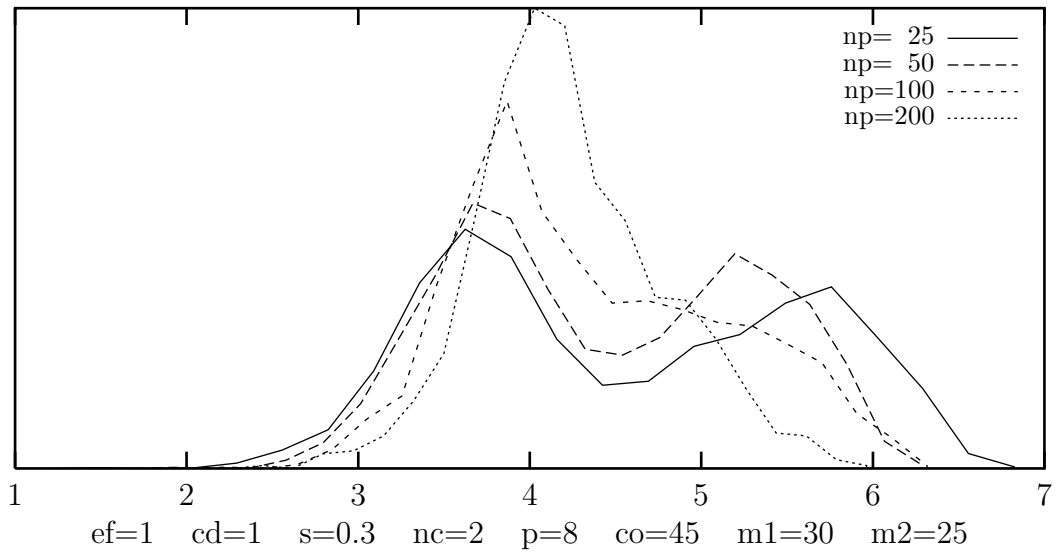


Figure 2: Dependency on the size of the population, evaluation function 1

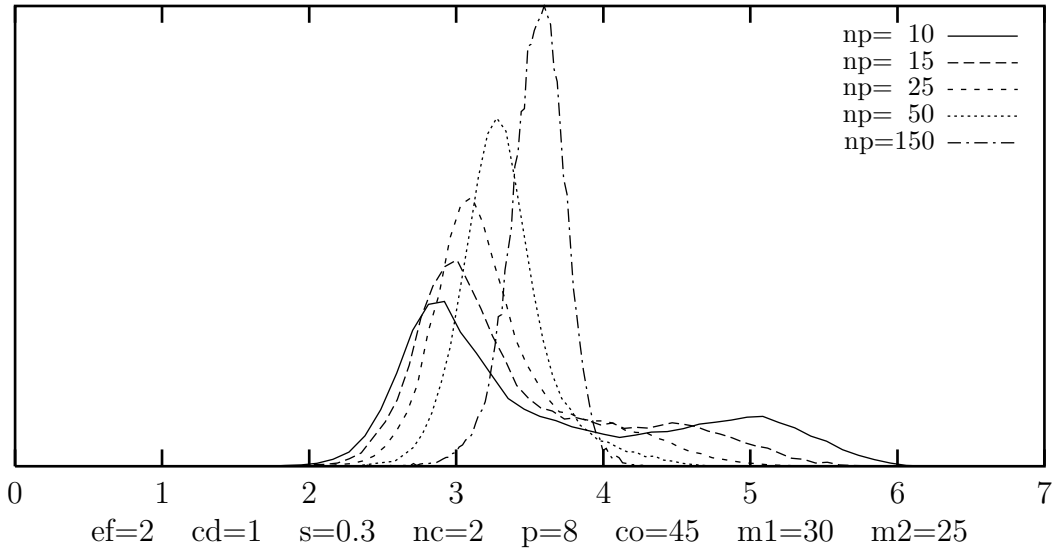


Figure 3: Dependency on the size of the population, evaluation function 2

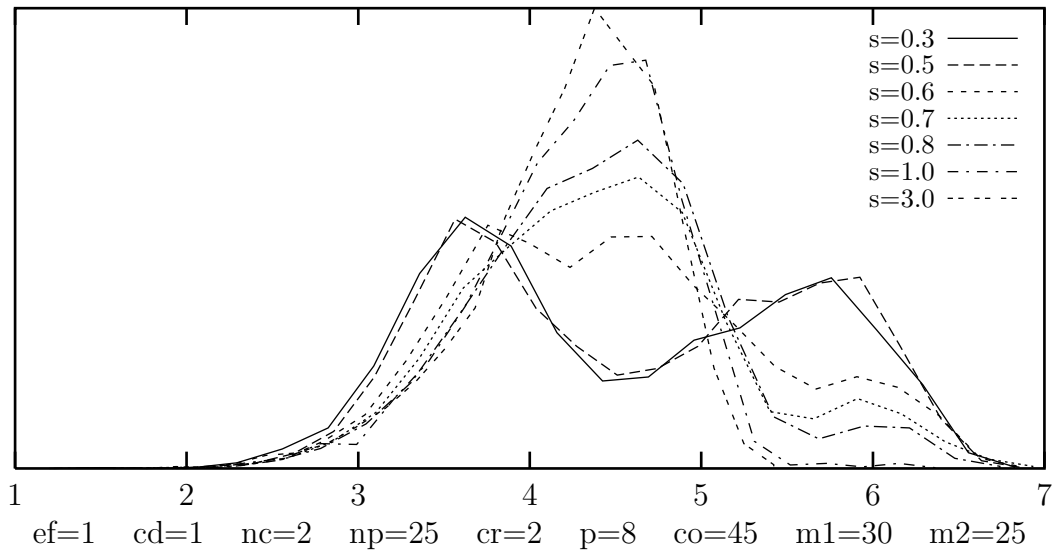


Figure 4: Dependency on the selectivity, evaluation function 1

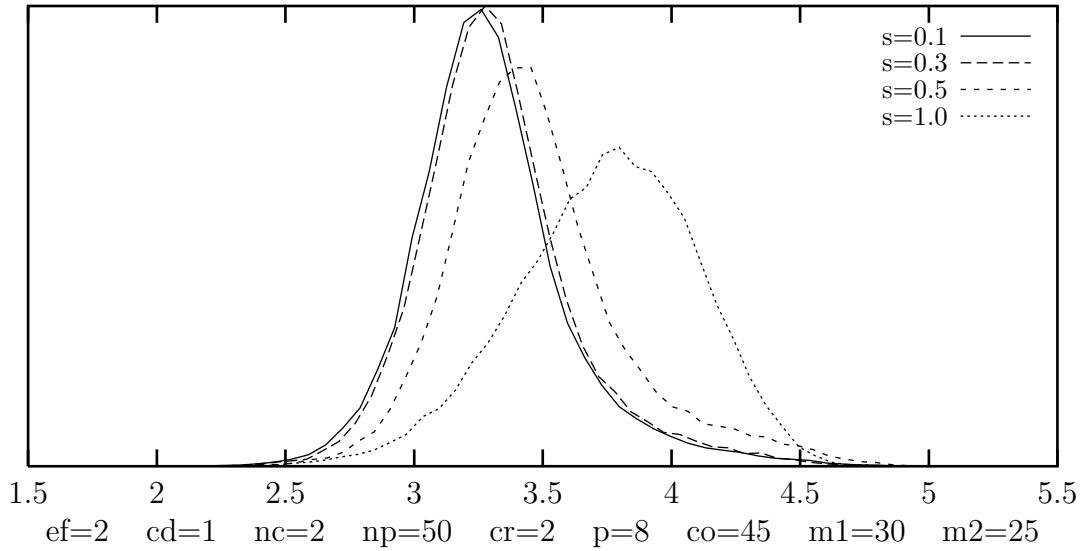


Figure 5: Dependency on the selectivity, evaluation function 2

## 6.4 Comparison of encodings

In figure 6 the two encodings `cd=1:` HFAGEKDBJC and `cd=2:` ABCDEFGHJK are compared for all three evaluation functions. It appears that grouping causally stronger depending genes next to each other in the chromosome, like in HFAGEKDBJC improves effectivity. This is the more effective the harder the problem is (or equivalently the less information the evaluation function provides) with the consequence that for the perfect evaluation function `ef=3` there is virtually no difference in the graphs for both encodings.

For the improved encoding `cd=1:` HFAGEKDBJC it is better to inherit bigger blocks

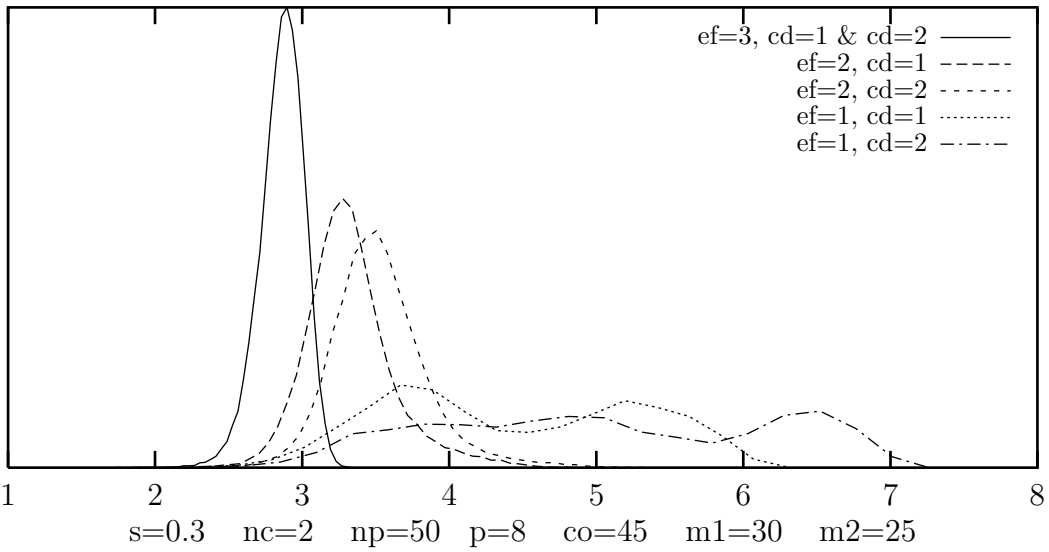


Figure 6: The two encodings with all 3 evaluation functions

from parents than for encoding 2. Although the dependency on the related inheritance-switching-probability $p$ in (1) is remarkably small, optimal efficiency is obtained for $p = 15 - 50\%$ for encoding 1 (figure 7) and $p = 50 - 70\%$ for encoding 2 (figure 8).

A change in the encoding has also an effect on the optimal ratio of crossover/mutation (= co/(m1+m2)). As can be seen from figure 9, for the encoding `cd=2:` ABCDEFGHJK crossover is useless and reducing its weight is the best one can do. Differently for encoding `cd=1:` HFAGEKDBJC for which about 55% crossover is the optimum (figure 10).

(These numbers have to be seen in the context of the maximal-crossover-repeats parameter `cr=2` which means that the ratio (crossover/mutation) starts as 55:45 but later in the evolution when genetic material narrows, the weight of mutation increases slightly.) It is remarkable that for the order based encodings used here, crossover only becomes productive if causally linked genes are grouped next to each other in the chromosome.

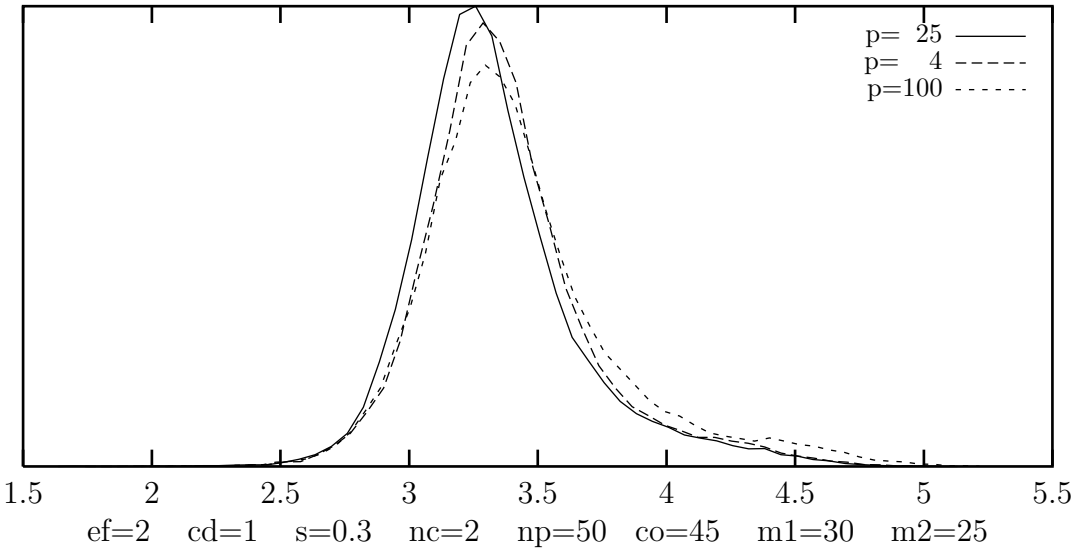


Figure 7: Variation of the switching probability in crossover, encoding 1

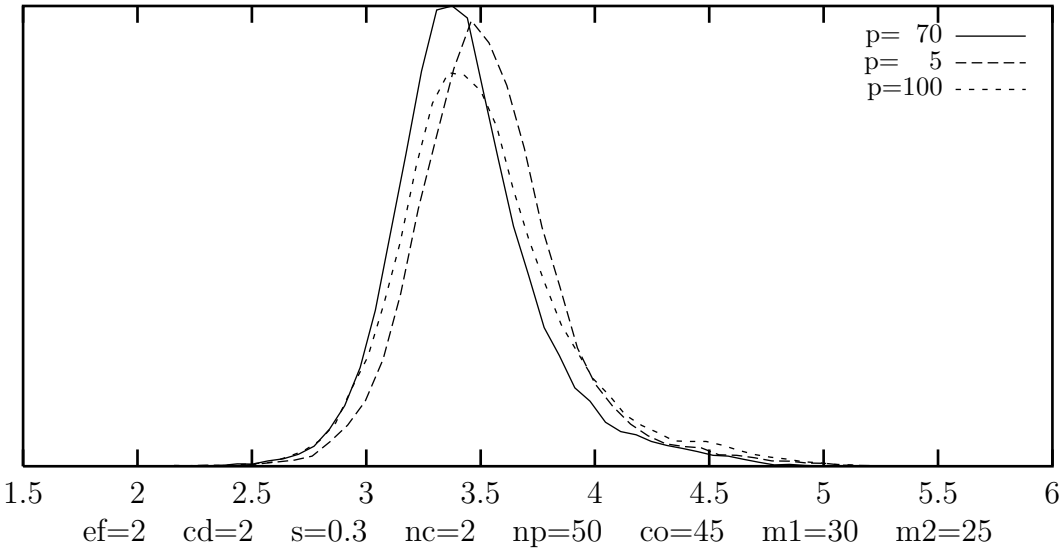


Figure 8: Variation of the switching probability in crossover, encoding 2

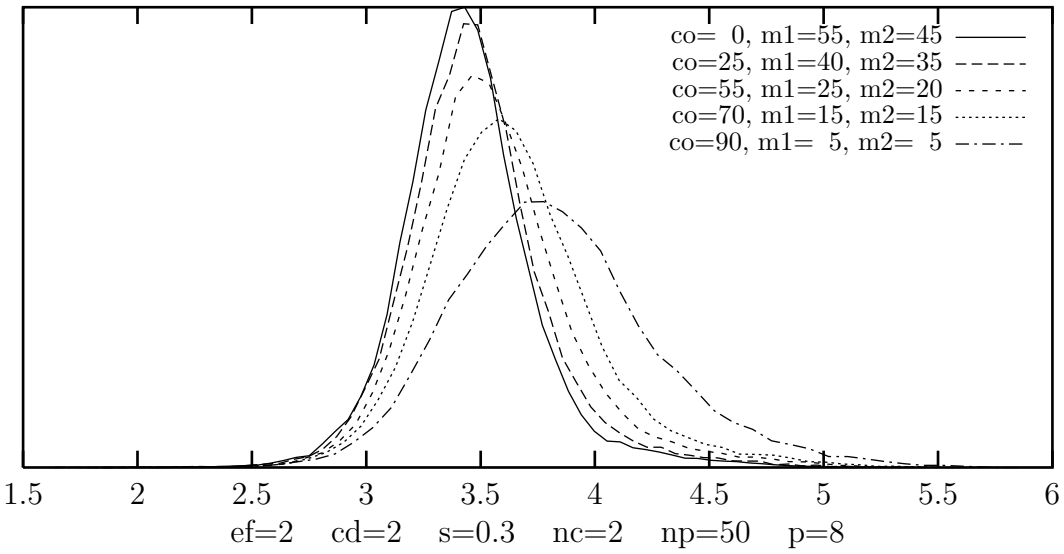


Figure 9: Variation of the ratio crossover/mutation, encoding 2

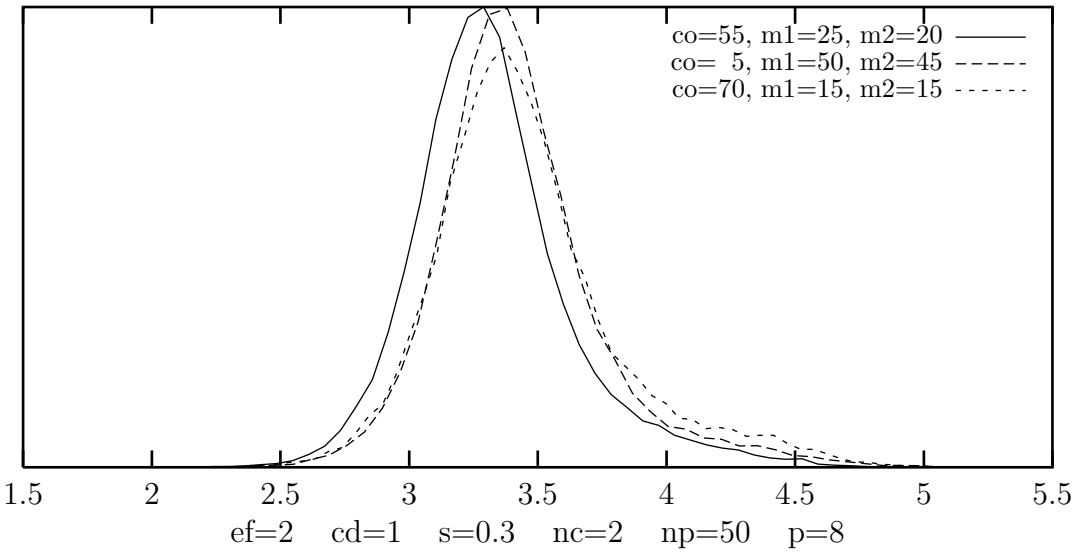


Figure 10: Variation of the ratio crossover/mutation, encoding 1

As this is a general principle, the author expects that, for example, for the Traveling Salesman problem it helps as well to position cities to be visited that are closely to each other geographically as well closely to each other in the chromosome.

## 6.5 The number of repeats of crossover

As described in section 3, offsprings generated through crossover are compared with all chromosomes in the population to avoid duplicates. In case of a clash, crossover is repeated `cr` times and then mutation is used instead to generate the next pair of offsprings. In figure 11, curves are plotted for `cr=100` and four different ratios crossover/mutation. In figure 12, curves for the same crossover/mutation ratios are plotted, there for `cr=1`. For the, about optimal, crossover/mutation ratio of 45/55 the difference in `cr` does not matter as the solid lines of both figures are comparable.

Differently for high values of `co`. The dotted line `co=96` in figure 11 is left shifted compared to the line `co=96` of figure 12. This means, when crossover already has an unfavourable high weight of 80-100% then in case of a clash of a new chromosome with an existing one, it is still better to repeat crossover as often as necessary until a new different chromosome is generated (dotted curve `co=96` in figure 11) instead of using mutation (dotted curve `co=96` in figure 12). The reason for this effect seems to be that the evaluation function used in figures 11, 12 is smooth enough such that it is better to continue using crossover and stay close to the maximum found so far.

The situation changes very much for the less smooth and low information evalua-

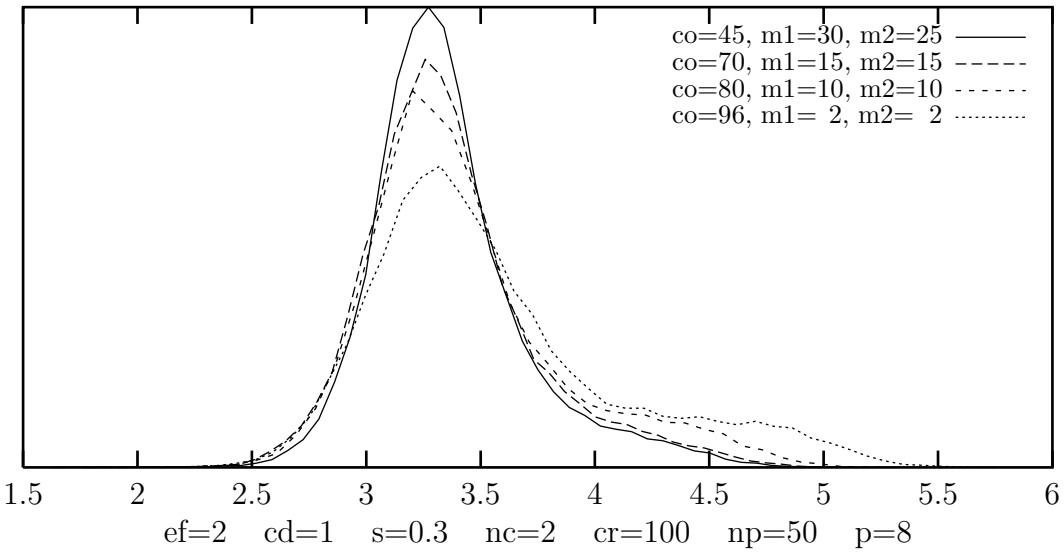


Figure 11: Many repeats of crossover, evaluation function 2

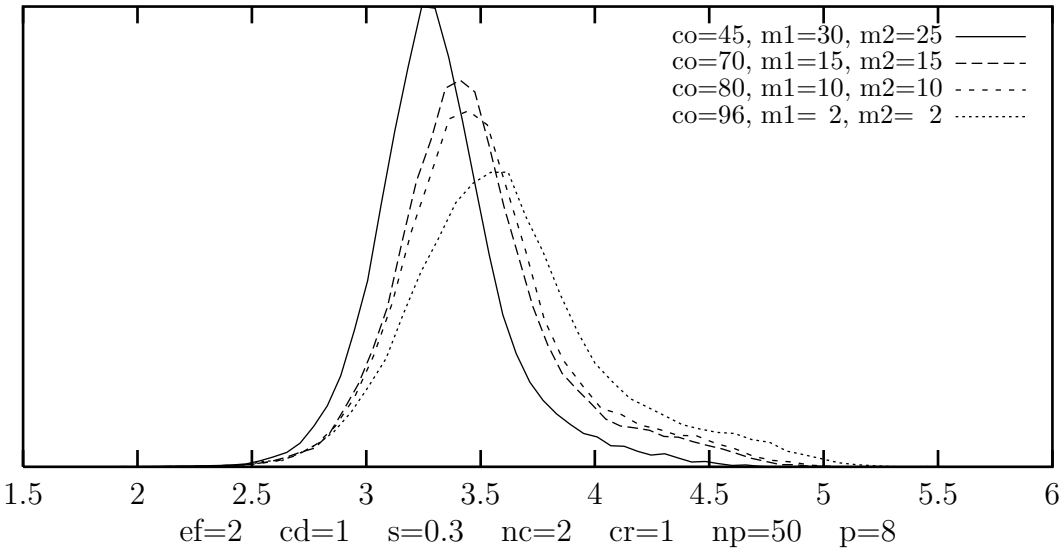


Figure 12: No repeats of crossover, evaluation function 2

tion function 1. In figures 13, 14 the same crossover/mutation ratios are used as in figures 11, 12.

The first difference is that no repeats of crossover (figure 14) is superior to many repeats of crossover (figure 13) as the right flanks in figure 14 are located further left than those in figure 13. This is in contrast to figure 11, 12 where the opposite is true. It was already indicated that the different nature of the evaluation function is responsible for that.

The second difference is rather remarkable. In figures 11, 12 and 13 a very high crossover weight of 96% is the worst of each of the four crossover/mutation ratios. Not so in figure 14 where the right flank of `co=96` is even left of `co=45`. The mechanism seems to be that for low information function 1 an initially high crossover weight of 96% does soon lead to a low genetic diversity, such that it comes earlier to extra mutations when an offspring generated by crossover is a duplicate of an already existing one.

## 6.6 Immortality of the best chromosome

As described in section 4, during the selection process the chromosomes with a top fitness are spared from being deleted. This is a good idea for a GA-soft problem but is it good as well for evaluation function 1 if one aims at low selectivity, i.e. deleting high scoring chromosomes as well?

In figure 15 two series of runs for the GA-hard evaluation function 1 are compared, one sparing top chromosomes, the other without special treatment for them,

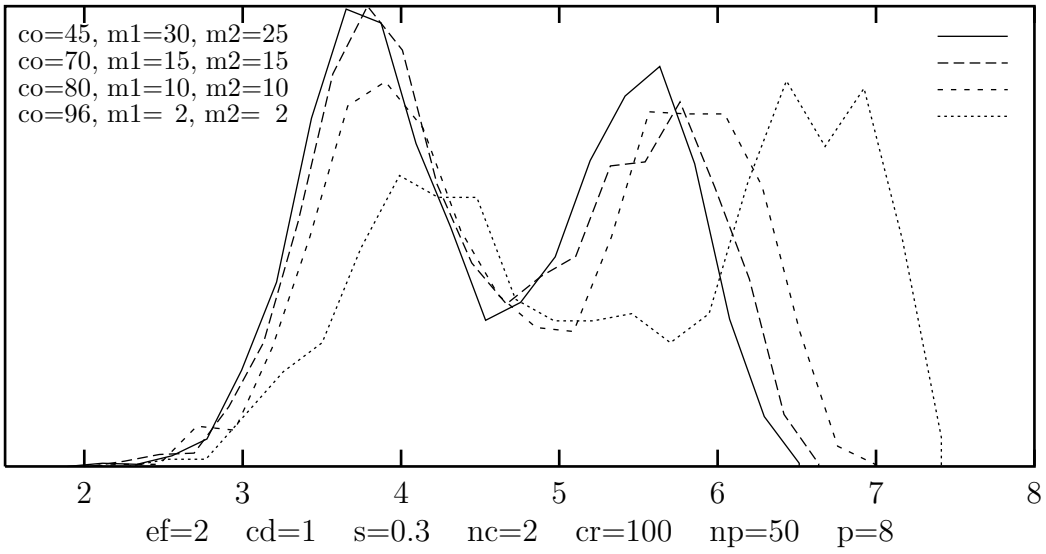


Figure 13: Many repeats of crossover, evaluation function 1

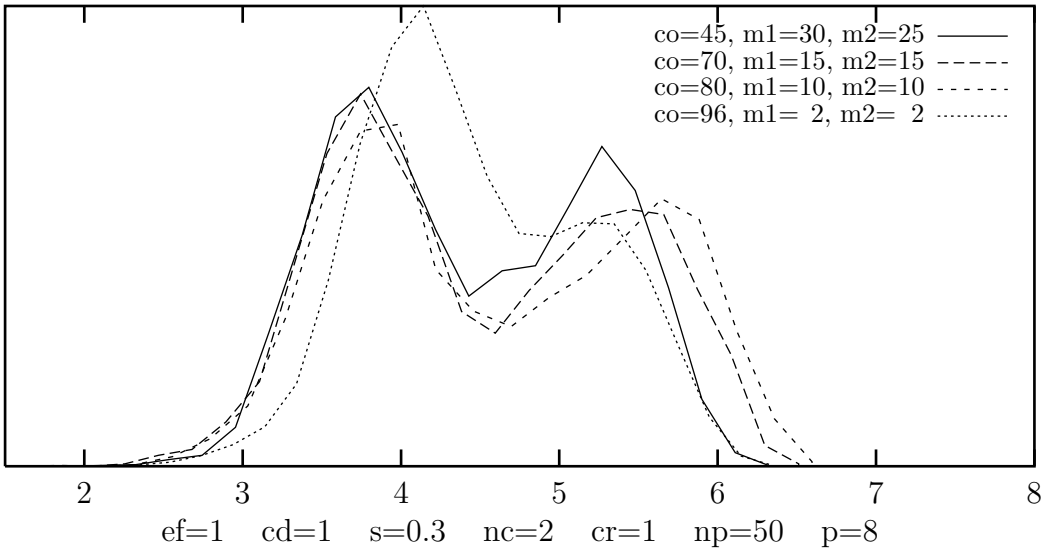


Figure 14: No repeats of crossover, evaluation function 1

i.e. deleting them if randomly chosen. Even for these parameters used here (`s=3`), the immortability of top chromosomes is superior.

Therefore, in case of evaluation functions with many local optima where selection must not be aggressive (high value of $s$), then as a counter measure, it obviously is useful to keep top chromosomes (unless all chromosomes have the same fitness) and rather have a higher value of $s$.

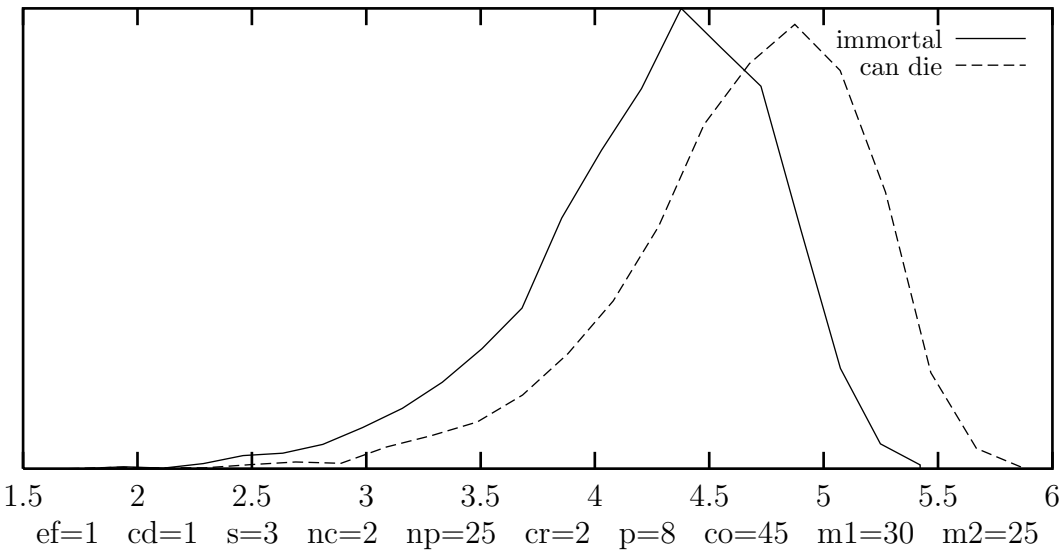


Figure 15: Immortality of top chromosomes

## 6.7 Automatically restarting evolution

Let us assume, the probability curve has two humps of roughly equal size separated by, say, 2 i.e. the second corresponding to a *factor* of $10^2 = 100$ more chromosomes generated, like in figures 13, 14. Then, as a single evolution progresses, if the number of chromosomes computed has passed the first hump without solving the puzzle then it is likely that about a factor of 100 more have to be generated, only to double the chance of finding the solution. But the same probability one can have much cheaper by restarting from scratch, thereby just doubling the effort. To include the option of restarting the evolution seems to be a cheat on the principles of the genetic algorithm. If the genetic material in the population has converged then something has to be done, for example, an increase of the weight of mutation. The restart of evolution is only the most radical of all steps towards more diversity. It is likely that other, less radical measures are better, if at least some of the genes of the best chromosomes are correct and worth inheriting.

In figures 16, 17 evolution is restarted if no improvement of the top fitness had been occured for many generations. We adopt a simple rule with one parameter `rf` (Restart-Factor).

Let $n$ be the number of generated chromosomes since the last start or restart. Let $m$ be the number of generated chromosomes from the last start or restart until the last time that a new record fitness was reached. The evolution is restarted as soon as $n > rf \times m$. In the case of a restart the complete population is randomly initialized, $n, m$ are set to $n = m =$ (size of the initial population), and the record fitness to be improved in the following is the highest fitness in the initial population. The vertical axis in figures 16, 17 below measures the total of all individuals generated in all restarts until the puzzle was solved.

The technique to restart if the evolution got frozen has the effect of shortening the longest runs, i.e. of shifting the right flank of our diagrams towards the left. In that it is similar to reducing selectivity (increasing the `s` value, figure 4) or increasing the size of the population (increasing the `np` value, figure 2). But low selectivity means that chromosomes near the global optimum may be deleted and early solutions are less likely - the left flank moves towards the right. Similarly, in a large population it takes longer for top chromosomes close to the global optimum to dominate the population and to lead to a quick solution of the puzzle.

Differently the restarting technique which has little effect on the left flank as this corresponds to situations where evolution is running well and improving fast. To have the left and right flank left-shifted as much as possible, we combine high selectivity and/or low population sizes to have many early solutions with the restart technique to have an insurance against conversion to a local optimum. This interpretation is well confirmed in figure 16 where the optimal curve with `rf=6` has the same left flank as the low-`s` curve and even improved on the right flank of the high-`s` curve. For evaluation function 2 the situation is different. From figure 5 we learned that high selectivity (low `s` values) are not beneficial which leaves only low population sizes to left-shift the left flank (figure 3 above) and use the restarting technique to improve on the otherwise right-shifting right flank. Because evaluation function 2 is so well behaved one has to go to very low population sizes to develop a second hump which could make the restarting technique useful. As it is seen from figure 17, although the very long runs are avoided, the right flank of runs without restart is not really improved. Therefore, as it is to be expected, the restart technique is more useful for GA-hard problems (evaluation function 1) where convergence to local optima happens more frequently than for GA-soft problems (evaluation function 2).

Another difference is that evaluation function 2 can have more different values and consequently the top fitness increases more frequently in smaller steps and therefore the restart factor has to be smaller. This restart mechanism can only be a rough example and a more differentiated increase of the number of children `nc` and the weight of mutation `m2` might be better instead of deleting the whole population.

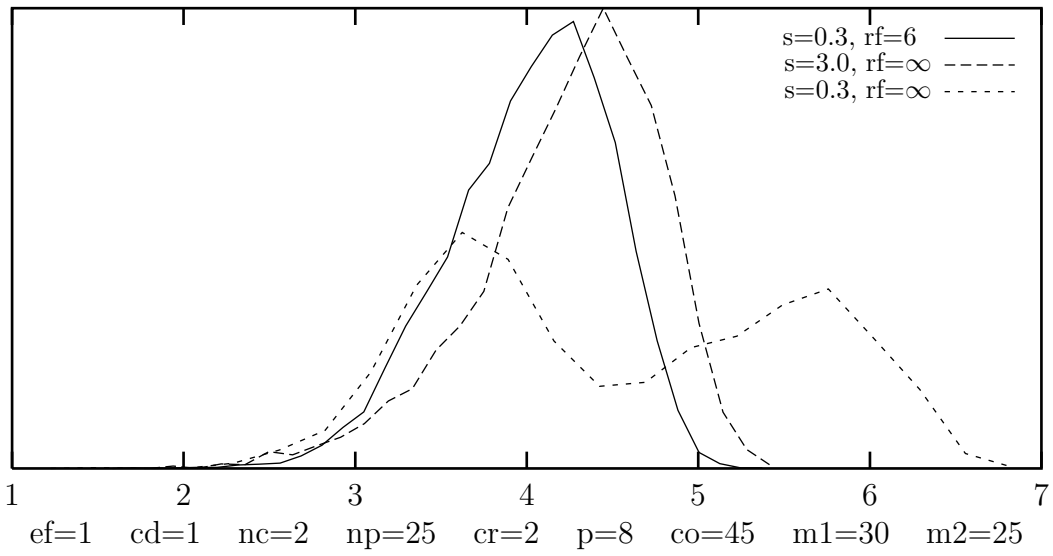


Figure 16: Restart technique, evaluation function 1

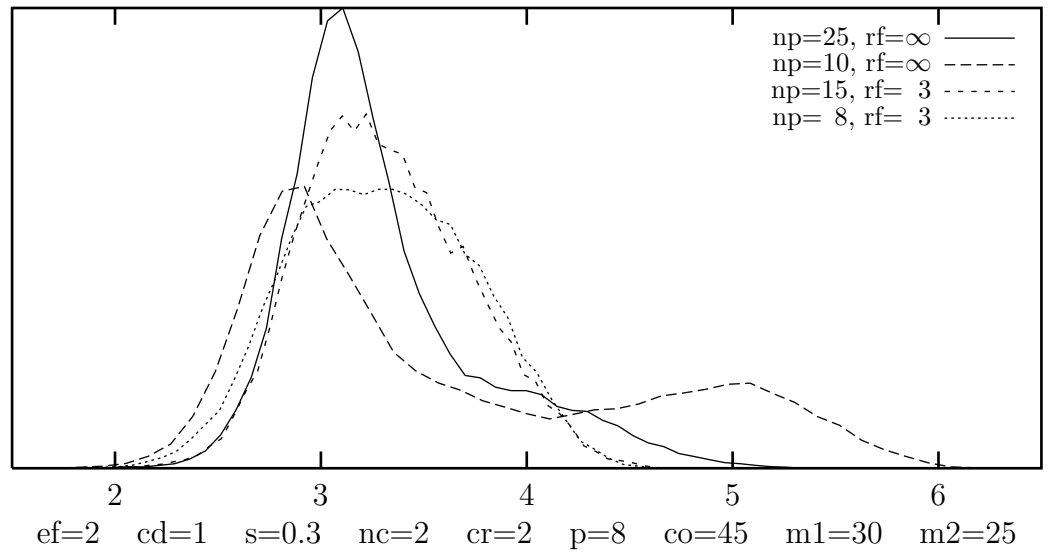


Figure 17: Restart technique, evaluation function 2

# 7 Summary

We found the simple but nontrivial puzzle to be useful in demonstrating many typical parameter dependences of the GA including the dependence on the choice of the encoding and the fitness function.

We found a strong relationship between the fitness function being GA-hard or not and the optimal size of the population, the optimal selectivity and the optimal restart factor.

Another relationship proved to exist between the encoding and the crossover / mutation ratio. Positioning causally stronger connected genes next to each other in the chromosomes increased the optimal crossover / mutation ratio and the optimal size of lumps inherited from parents during crossover.

Interestingly, in GA-hard situations the probability curve showed two maxima (on a logarithmic horizontal axis) which suggests the following interpretation.

Either during evolution there are enough chromosomes generated that are in reach of the global optimum (through a large population or low selectivity and enough mutation) and then the puzzle is solved quickly, otherwise a lower local optimum will be found instead and then it will take very long to leave it.

Usually combinatorial problems of interest have a vastly bigger configuration space than ours of only 10! permutations. In that case the ratio (size of population)/(number of configurations) will inevitably have to be much smaller. Measures to overcome this are

- a large population (big `np`),
- a low selectivity (big `s`),
- switching to strong mutation (or increasing the weight of mutation) if crossover produces offsprings that are already in the population,
- restarting evolution if no progress is reached after a longer interval.

# 8 Possible extensions

In all investigations above the configuration space and therefore its size was constant. Although the trend of parameter dependence for an enlarged configuration space is clear, it would be nice to see it explicitly. A way to do this for a similar problem would be to solve a puzzle like the one in this paper but with letters corresponding to digits not in the interval $0 \ldots 9$ but, for example, $0 \ldots 15$. Such puzzles to a different base are not known to the author and would have to be invented first, also using Genetic Algorithms.

Another extension would be to solve the above puzzle with other optimization techniques like Simulated Annealing, each with varied parameter values, and to compare them with the best runs of the Genetic Algorithm.

# References

[1] Davis, L. 1991. Handbook of Genetic Algorithms. New York: Van Nostrand Reinhold.

[2] Goldberg, D.E. 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MAA: Addison-Wesley.