Paper 9:

Genetic algorithm approach for the three-matching problem

Gábor Magyar, **Mika Johnsson**, **Olli Nevalainen**
Turku Centre for Computer Science (TUCS) and
University of Turku, Department of Computer Science
Lemminkäisenkatu 14, FIN-20520 Turku, Finland
E-mail: `olli.nevalainen@utu.fi`

Abstract.

In this paper we discuss the solution of the Three-Matching Problem (3MP) by a genetic algorithm (GA). In the 3MP we want to partition a set of $n = 3l$ points into l disjoint subsets, each consisting of three points (called triplets), and to minimize the total cost of the triplets. We concentrate on the special case where the cost of a triplet is the Euclidean edge length of the minimal spanning tree of the three points.

The problem belongs to the class of hard (NP-complete) graph problems and has many practical applications in electronic industry (e.g. the optimization of manual operations of component insertions). The individuals of GA are coded as permutations of the points. We give two new crossover operators, the triplet union and edge union crossover operators. Our GA is fully parametrized so that the user can choose its operators which are applied to different generations. Our empirical tests show that the GA outperforms such heuristic methods as simulated annealing and local search, which we have studied previously.

Keywords: genetic algorithms, combinatorial optimization, graphs, hill climbing, matching problems, TSP

9.1 Introduction

This paper discusses the *Three-Matching Problem* (3MP): Given a complete weighted graph of $n = 3l$ points, we search for a partition of this graph into

l disjoint subgraphs so that each subgraph is a *tree* of three vertices and the total cost of the edges in the trees (called here *triplets*) is minimal. The 3MP was studied in [314]. That paper summarizes several heuristic approaches to solve the 3MP, including genetic algorithms, local search methods and simulated annealing. Here we discuss the genetic algorithm (GA) approach in greater detail.

We focus on solving the *Euclidean* 3MP, where the graph points are in the 2-dimensional space and the edge weights are the Euclidean distances. However, the described GA can solve general 3-matching problems, too.

It has been shown that the 3MP is NP-complete [314]. The same paper gives also several integer programming formulations and lower bounds for the problem. Although the 3MP is related to the Travelling Salesman Problem (TSP), their solutions are very different.

The 3MP originates in an industrial application for sequencing the operations of inserting electronic components on a printed circuit board. In this application we have to divide a set of points into triplets and traverse them in a minimal way, see [315] for more information. Figure 9.1 shows a 51-point instance of 3MP.

9.2 The GA model

This section describes the GA architecture which we use for solving the 3MP. First, we present the general GA framework and discuss its components like the chromosome representation, the fitness function along with the selection, crossover and mutation operators. At the end of the section we summarize the various control parameters for the GA.

9.2.1 The basic GA cycle

We use a *generational* GA with fixed population size. At each generation, the better solutions, the *parents*, are selected by a *selection method* from the current population to create the offspring. The selection method uses a fitness function which measures the quality of the solution represented by that chromosome. Each pair of the selected parents will mate and create two child chromosomes by a *recombination* operation. After the recombination, a *mutation* may occur.

It is typical to generate the initial population randomly. The subsequent generation of the populations terminates either after a predefined number of iterations or if the best member of the latest populations has not improved during a certain number of iterations.

Our genetic algorithm includes several different selection and recombination operators. One can specify the desired operators by a *selection method descriptor string* and also similarly by a *recombination operator descriptor string.* This works as follows. Suppose that the selection method descriptor string is 'ABD'. This states that at the first generation, the selection method A is used, then,

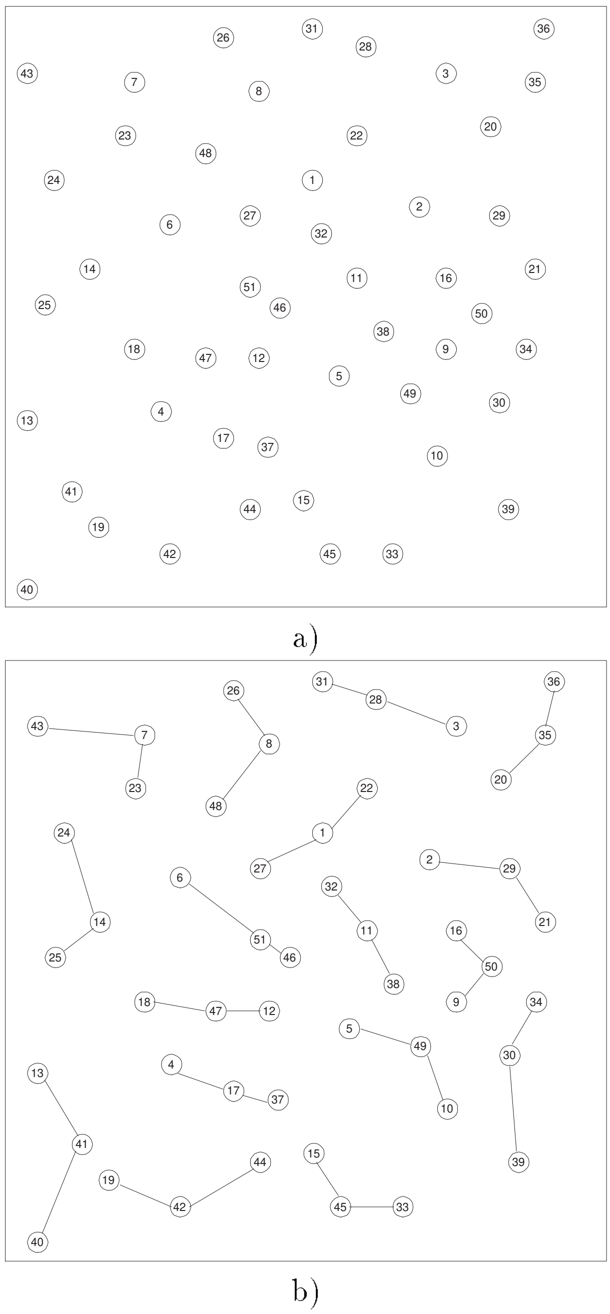

Figure 9.1: A sample 51-point 3MP problem instance (a) ('eil51' from TSPLIB [413]) with its optimal solution (b) [314]. The vertices are numbered as they appear in the problem file.

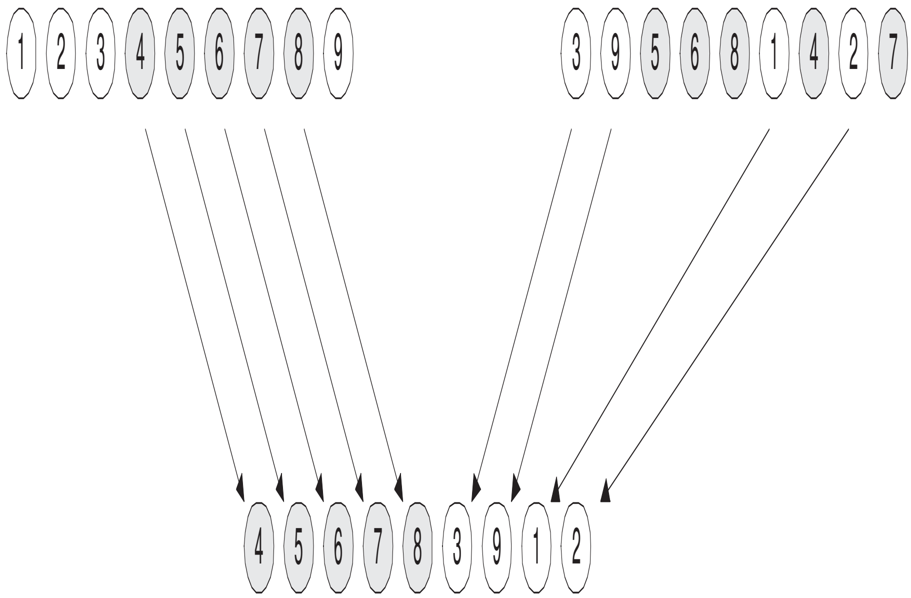

Figure 9.2: An example of the sorted-match crossover.

at the second generation, method B is chosen, the third generation uses the method D, and then, at the fourth generation, method A is used again, cyclically. The recombination operator is specified similarly. Note that the length of these descriptor strings is arbitrary, and that a descriptor string can contain a certain character (i.e. a reference to a method) more than once.

In the following, we first examine the chromosome representation used to encode the feasible 3MP solutions, and then describe the individual selection and recombination methods.

9.2.2 Representation

In order to build a GA for any optimization problem, one must code the feasible solutions of the search space into *chromosome* format, i.e. a string of symbols. The genetic search is carried out by manipulating these chromosome representations.

Theoretically, the most suitable encodings use only 0/1 bits in the chromosome. Unfortunately, this binary coding is not effective for many combinatorial optimization problems, like for the TSP or 3MP. These problems usually use some kind of permutation representation.

For the 3MP, we use a permutation representation. A coding of a solution enumerates the triplet points in such a way that the middle point of the triplets comes always as the second. Every consecutive three-point subsequence uvw describes a triplet, where there are edges between $u - v$ and $v - w$. To illustrate the coding, consider the solution showed in Figure 9.1, which can be represented as the following string:

$$(43\ 7\ 23 \ \mid \ 26\ 8\ 48 \ \mid \ \ldots \ \mid \ 34\ 30\ 39)$$

Note that the order of the individual triplets is not important, therefore a certain solution has a large number of different chromosome representations. This kind of redundancy in the representation is still acceptable because the above coding is well suited for the genetic operations used here. Moreover, the triplets can be listed in two ways, e.g. the first triplet in the above chromosome can as well be coded as 23 7 43. This gives us in total $2(l!)$ different representations of a particular solution. We apply a *improvement operator* after every child creation. This operation rearranges the three points in the triplets optimally (i.e. we simply put the two farthest points to the edges of a triplet).

9.2.3 Fitness functions

The three available fitness functions are the following:

$$\begin{aligned} f_i &= f_{\max} - 0.5 \cdot (z_i - z_{\min})/(z_{\max} - z_{\min}) \\ f_i &= z_{\max} - z_i + 1 \\ f_i &= (z_{\max} - z_i + 1)^2 \end{aligned}$$

where f_i denotes the fitness of the i^{th} individual, $f_{\max}$ is a constant (the desired fitness for the best member of the population), z_i denotes the length (objective function value) of a solution, $z_{\min}$ is the length of the best individual in the current population, while $z_{\max}$ stands for the length of the worst individual in the current population.

Function (9.1) performs a scaling so that the best solution will have twice the fitness as the worst. The other two functions perform dynamic scaling, where the worst individual of the population has the fitness value 1, while the fitness of the other individuals depends on their distance from the worst individual.

9.2.4 Selection schemes

Next we will briefly recall four well-known selection techniques that can be used in our GA.

Proportional Selection. In the well-known *roulette-wheel selection method* [170] the probability of selecting a certain chromosome for mating is proportional to its fitness value, i.e. the individual selection probabilities are the following:

$$p_i = \frac{f_i}{\sum_{i=1}^{\text{popsize}} f_i}$$

The selection is carried out by turning an artificial roulette wheel, where each slice represents an individual, and the sizes of these slices are proportional to the fitness values of the individuals.

Tournament Selection. The *tournament selection* [170, 338] provides an effective and robust way for selection. First we draw k individuals from the population and then select the best among these. The value of k is a parameter which controls the selective pressure. The way we select the k individuals at the first stage can be simply random or the roulette-wheel selection method described above.

Rank Selection. The family of *rank selection* methods [183] calculates for each individual i a *sampling rate* based on its rank in the population. We have implemented two methods; a *linear rank selection* and a *non-linear rank selection.* The calculation of the sampling rates is carried out by the following formulae:

$$\begin{aligned} n_i &= \mathrm{MIN} + (\mathrm{MAX} - \mathrm{MIN}) \frac{\mathrm{popsize} - \mathrm{rank}_i}{\mathrm{popsize} - 1} \\ n_i &= K \cdot \sqrt{\mathrm{popsize} - \mathrm{rank}_i + 1} \end{aligned}$$

where rank_i is the rank value (between 1 and popsize, $\mathrm{rank}_i = 1$ for the best), MIN is the desired minimal (for the worst member of the population) and MAX is the maximal (for the best member of the population) sampling rate. One can see that in order to have popsize as the sum of individual sampling rates, $\mathrm{MIN} = 2 - \mathrm{MAX}$ and $1 \leq \mathrm{MAX} \leq 2$ must hold, so only the value of MAX is an additional parameter to this method. (Note that if $\mathrm{MAX} = 1$, then all population members will be selected exactly once.) In the non-linear rank selection, the value of K is chosen in such a way that the sum of the individual sampling rates equals to popsize, i.e. $K = \mathrm{popsize} / \sum_{i=1}^{\mathrm{popsize}} \sqrt{i}$. An example of the above calculations with $\mathrm{popsize} = 6$ is shown in Figure 9.3.

Best Individuals Selection. The last group of the selection methods we call the *best individuals selection.* These methods always select the best individuals of the population in some way. Thus we in fact perform a kind of local search and this may lead to a loss of the diversity in the genetic search.

In the *best&random* selection, one parent is always the best individual of the actual population, while the other one is selected randomly.

	n_1	n_2	n_3	n_4	n_5	n_6
`linear`	2.00	1.60	1.40	0.80	0.40	0.00
`nonlinear`	1.36	1.24	1.11	0.96	0.78	0.55

Figure 9.3: Sampling rates for a population of 6 members. The value of MAX is 2 for the linear rank selection.

In the *best-k&random* selection, one of the parents is taken randomly from the first k best members of the actual population, while the other one is totally random. The value of k is now a parameter.

Finally, in the *best-k* selection, both parents are picked from the k best individuals of the actual population. The value of k is selected so that the number of all possible pairs matches the population size, i.e. $k = \lfloor\sqrt{\text{popsize}}\rfloor + 1$.

9.2.5 Recombination operators

Here we briefly describe four *crossover* operators for creating the offspring from a pair of parents. A group of the crossover operators includes general operators designed for problems with permutations, c.f. TSP. It has been shown that including problem-specific knowledge in the GA is often important in order to get high quality results. Therefore the other group uses some problem specific information.

Partially Matching Crossover (PMX). The operator called *partially matching crossover* (PMX) [170] is a kind of 2-point crossover tailored to the representation of permutations. We choose two crossover points randomly, and then exchange the contents between these crossover points in the two parents. In order to have two permutations as a result, we must do an additional repair on the resulting chromosomes. When making these additional exchanges, the PMX tries to preserve the parents' absolute point positions in their children. The mechanism behind the PMX is illustrated in Figure 9.4.

$$
\begin{array}{l}
A = 9\ 8\ 4\ |\ 5\ 6\ \ 7\ |\ 1\ 3\ 2\ 10 \\
B = 8\ 7\ 1\ |\ 2\ 3\ 10\ |\ 9\ 5\ 4\ \ 6 \\
\qquad\qquad\Downarrow \\
A^* = 9\ 8\ 4\ |\ 2\ 3\ 10\ |\ 1\ 3\ 2\ 10 \\
B^* = 8\ 7\ 1\ |\ 5\ 6\ \ 7\ |\ 9\ 5\ 4\ \ 6 \\
\qquad\qquad\Downarrow \\
A' = 9\ \ 8\ 4\ |\ 2\ 3\ 10\ |\ 1\ 6\ 5\ 7 \\
B' = 8\ 10\ 1\ |\ 5\ 6\ \ 7\ |\ 9\ 2\ 4\ 3
\end{array}
$$

Figure 9.4: Partially matching crossover. A, B are the parents, A^*, B^* are the intermediate chromosomes after the exchange, while A', B' are the legal child chromosomes after the repair. The mappings 5-2, 3-6 and 10-7 were used at the repair.

In addition to the original PMX, we have also included two modified versions of it. In the PMX-3mp, we demand that the two crossover points must be exactly at triplet borders, i.e. the possible crossover positions are positions 1,4,7,...for the 1st crossover point and positions 3,6,9,...for the 2nd crossover point. In the PMX-prep, we select the crossover points as in the PMX-3mp. Additionally, before selecting the crossover points, we match the two chromosomes so that similar triplets are at equivalent places in both strings. This is done by passing through the triplets of the second parent once and finding the best matches with the triplets of the first parent (in a greedy way). By this we try to avoid most of the repair operations at the second phase of the PMX. For the idea of the preprocessing, see Figure 9.5.

$$
\begin{array}{l}
A = 1\ 2\ 3 \mid 4\ 5\ 6 \mid 7\ 8\ 9 \\
B = 5\ 4\ 9 \mid 1\ 3\ 6 \mid 2\ 7\ 8 \\
\qquad\qquad \Downarrow \\
A' = 1\ 2\ 3 \mid 4\ 5\ 6 \mid 7\ 8\ 9 \\
B' = 1\ 3\ 6 \mid 5\ 4\ 9 \mid 2\ 7\ 8
\end{array}
$$

Figure 9.5: The preprocessing of parents A, B will lead to parents A', B'. Note that the triplets of the second parent are moved only.

Sorted Match Crossover (SMX). In the *sorted match crossover* (denoted here SMX-3mp), we first choose a random substring from one of the parents called a *donator*. The first child chromosome will begin with this string segment. We fill the remaining part of the first child chromosome from the other parent (it is called here the *receiver*) as follows. We first delete the digits of the chosen substring, then copy over the rest. This gives us a permutation again. We demand that the substring picked from the donator starts (and ends) at a legal triplet starting (ending) position as in the PMX-3mp. The second child is created similarly, but starting from the parent which was the receiver at the above case. Figure 9.2 illustrates this operation. The SMX-3mp is similar to the Order Crossover (OR), see [346].

Triplet Union Crossover (TUX). The *triplet union crossover* (denoted by TUX) is a knowledge-based operator that tries to recombine the existing triplets from two parents into their offspring.

The crossover works as follows. Suppose we have parents A and B. First, we create a list of all the triplets included in A and B. This list will contain initially $2n/3$ triplets, where n denotes the problem size. After creating this list, we sort its elements into an ascending order by the length of the triplets. The two child chromosomes are generated by the same method which works as follows.

1. Let the child chromosome be initially an empty string.
2. *(Iteration)* Select the shortest triplet of the list (denoted by uvw) and add it to the child chromosome.
3. Remove from the sorted list all triplets which include at least one of the points u, v or w. If the list is non-empty, go to the next iteration step.
4. If the child is not complete, fill the remaining part with the unused points by a heuristic or random method.

The filling of the remaining parts of the child chromosomes can be totally random (*TUX-rnd*) or it can be made by a greedy heuristic (*TUX-heu*). In the latter method we select a point u randomly from the set of remaining points. We then search for the closest point to this point from the set of remaining points, let us denote it by v. Similarly, we search for a third point w from the set of remaining points for this $u - v$ edge to form a triplet in such a way that this additional edge ($w - u$ or $v - w$) has a minimal length. Finally, we add the resulting triplet (wuv or uvw) to the child chromosome and continue with another iteration if the set of the remaining points is still non-empty. (This greedy heuristic method provides some kind of random choice which is necessary for maintaining the genetic diversity. However, it is quite restricted, and the other disadvantage is that the first segment of the two child chromosomes is identical, since they use the same method for creating that part.)

Edge Union Crossover (EUX). The *edge union crossover* (EUX) is also a kind of knowledge-based crossover. This operator works similarly to the TUX, but now the edges are recombined from the parents.

The crossover works as follows. First, we create a list of all the edges included in parents A and B. This list will contain initially $4n/3$ edges. We sort the list into an ascending order by the length of the edges. The two child chromosomes are generated as follows.

1. Let the child chromosome be initially an empty string.
2. *(Iteration)* Select the shortest edge of the list and denote it by uv. Delete the uv edge from the edge list.
3. Find the edge which includes u or v and its length is minimal. Let w be the other end point of that edge. (If there are no more edges with the points u and v in the list, then select a third point w in such a way that w is present in some of the edges of the list, and $\min(d_{wu}, d_{vw})$ is minimal.)
4. Add the resulting triplet (wuv or uvw) to the child chromosome.
5. Remove from the sorted list all edges which include at least one of the points u, v or w. If the list is non-empty, return to the next iteration step.
6. If the child is not complete, fill the remaining part of the child with the unused points by a heuristic or random method.

The filling of the remaining parts of the child chromosomes – as in the above triplet union crossover – can be totally random (*EUX-rnd*) or it can be made by the same greedy heuristic (*EUX-heu*).

9.2.6 Mutation

Mutation (MUT). The *mutation* operator is commonly a vital component of a GA. The recombination operators are responsible for exploiting a certain region of the search space, while the mutation has its role in exploring the regions of the search space.

We apply *regular mutation* for each new individual with a small probability (p_m) and a *major swap* for all the population. The GA will perform a complete mutation on all population members if the best member of the latest populations has not improved during the last m generations.

To mutate a chromosome C, we first select the number k of positions to be exchanged. The number k is chosen randomly from 3 to *mmax*, where *mmax* is another parameter. We pick randomly k positions $(i_1, \ldots, i_k)$ in the string and shift the contents of those string positions, so $C_{i_1} = C_{i_2}, C_{i_2} = C_{i_3}, \ldots, C_{i_k} = C_{i_1}$.

Inversion (INV). The operation selects a random substring of the chromosome in the argument and inverts it, e.g. if the string `1 2 | 3 4 5 6 | 7 8 9` is inverted between the marked positions, we get the string `1 2 | 6 5 4 3 | 7 8 9`.

Hill-climbing (HC). Evolutionary algorithms have many good properties, but it is usually not enough to use blind crossover operators to achieve the best results. Incorporating a kind of *hill-climber* (HC) is often important. For a given chromosome, HC performs a *2-opt* operation; at each step, two selected triplets are compared to see whether a new point allocation could yield better triplets. The triplets to be compared are drawn in a specific order. The 2-opt terminates if it is not possible to improve the solution by comparing any two triplets. For more details about the 2-opt for the 3MP, see [314].

9.2.7 Summary of the GA parameters

The control parameters of the GA are the following:

1. General GA parameters include the *population size*, the maximal *generation number*, and the *stopping criterion*, i.e. the number of generations when the best member of the latest generations has not improved. If the stopping criterion is not set, then the GA will iterate all the specified generations.

2. The type of fitness function used to evaluate the chromosomes.

3. The *selection descriptor string* can include the following methods: *RW* (proportional selection by roulette wheel), *k-TOUR-rnd* (tournament selection with size k, randomly selected candidates), *k-TOUR-rw* (tournament selection with size k, candidates selected by roulette-wheel), *LRS-#max* (linear rank selection, *max* is a parameter between 1 and 2), *NRS* (non-linear rank selection), *best&random*, *best-k&random* (k is a parameter) and *best-k*.

parameter	'A'	'B'	'C'	'D'	'E'
popsize	500	500	200	10	24
maxgen/stop	500/50	500/40	500/50	100/10	500/30
fitness	2	2	2	2	2
selection descriptor	5-TOUR-rnd	4-TOUR-rnd	LRS-2; NRS	best-k; LRS-1	RW; 2TOURrw; RW; LRS-1; best-k; LRS-1
recombination descriptor	PMX-prep; SMX; INV	TUX-rnd	EUX-rnd	MUT; HC	EUX-rnd; INV; EUX-heu; HC; MUT; HC
p_r	0.90	0.95	0.95	1.00	1.00
only_impr.	+	−	−	−	−
p_m	0.005	0.005	0.010	0.000	0.000
mmax	15	20	15	10	20
major swap	30	20	25	5	16

Table 9.1: The parameter settings included in the tests.

4. The *recombination descriptor string* can contain the following references: *PMX, PMX-3mp, PMX-prep, SMX, TUX-rnd, TUX-heu, EUX-rnd, EUX-heu, INV, HC, MUT.*

5. Two parameters are connected to the reproduction. The recombination probability (p_r) determines how many percent of the selected parents will mate to create the new generation. The other part of the parents $(1 - p_r)$ will go to the next generation unchanged. We will use $p_r = 0.9 - 0.95$. The other parameter is a boolean flag only_improve. If the flag is set, we allow only an offspring with an improved fitness value. This means that we create the two child chromosomes and select the best two chromosomes from the parents and their children, and these pass to the next generation. The flag has a similar effect as p_r.

6. The parameters concerning the mutations are p_m, *mmax* and the *major swap.*

9.3 Empirical analysis of the GA parameters

Table 9.1 illustrates some typical parameter settings included in our tests with the generalized GA. The parameter settings are labelled from 'A' to 'E' and they can be interpreted as follows. Setting 'A' is a traditional approach with general recombination operators for permutation representation. Settings 'B' and 'C' are using the problem specific recombination operators. Settings 'D' and 'E' are using the idea of parallel genetic hill-climbing. Setting 'E' differs from 'D' in a sense that it incorporates problem specific recombination operators between the hill-climbing steps.

Problem	'A'	'B'	'C'	'D'	'E'
p01	135.29	119.96	119.94	100.00	100.63
p02	145.25	118.70	115.36	100.00	100.24
p03	135.88	119.75	114.76	102.21	100.00
p04	138.34	125.26	118.82	101.65	100.00
p05	142.26	124.46	117.40	100.71	100.00
p06	134.02	123.20	114.97	100.31	100.00
p07	138.48	116.06	117.19	101.07	100.00
p08	134.75	120.45	114.03	100.86	100.00
p09	149.48	120.21	112.85	100.00	100.08
p10	137.62	117.86	122.26	100.58	100.00
average	139.14	120.59	116.76	100.74	100.09
# of best	0	0	0	3	7

Table 9.2: The performance of the different GA variants.

Table 9.2 shows the performance of the different GA variants. The results are expressed in percentages of the best result obtained. The results are for randomly generated problems of size $n = 201$. Here the results are for only one solution of the problem (variation of the solutions was very low for repeated runs of the same method). The last row indicates the number of times the given setting produced the best result.

The results show that the variants which use the HC operation are much better than the others. The traditional approach failed to produce acceptable results for our problem. Settings 'B' and 'C' performed better than 'A' but the last two settings with the iterative hill-climbing performed best. The variant which uses crossovers in between the hill-climbing-mutation loop (i.e. 'E') slightly improved the results of the pure hill-climber ('D'). Variant A is the fastest of the five methods. Variants B and C use approximately twice the time of A. Variants D and E are the slowest methods requiring ten times more calculation than A.

9.4 Comparison with other methods

Table 9.3 compares the GA with *simulated annealing* (SA) and *local search* (LS). The results are expressed in percentages of the best result found. The table contains results for 13 problems with the sizes between 84 and 1059. The problems are either random or picked from TSPLIB [413], and one of them is an industrial problem. The last column contains the length of the best solution and the last row shows the number of times the given method has found the best solution. The features of the SA/LS methods are described in [314].

In our tests GA usually gave better solutions than the other methods. Note

Problem	GA-'D'	GA-'E'	SA	LS	Best sol.
`manual84`*	100.00	100.00	100.00	101.12	1007.11
`rat99`**	100.61	100.00	100.49	101.83	753.87
`120a`***	100.01	100.00	100.35	101.83	1509.42
`krob150`**	100.00	100.00	100.67	100.68	15227.38
`ch150`**	100.68	100.00	100.33	102.18	3921.43
`201a`***	101.62	100.00	101.76	103.28	1945.09
`240a`***	101.80	100.00	101.12	102.57	2103.77
`300a`***	100.05	100.00	102.07	101.57	2221.04
`360a`***	100.00	100.53	101.02	100.62	2572.31
`510a`***	101.81	100.00	101.80	101.75	3081.35
`rat783`**	103.53	100.20	101.34	100.00	5458.59
`pr1002`**	104.38	100.00	104.62	102.98	151625.88
`u1059`**	104.06	100.61	101.54	100.00	134547.42
average	101.43	100.10	101.32	101.57	
# of best	3	10	1	2	

* Industrial application [315]
** TSPLIB [413] (Note: 'u1059' originally has 1060 points, its last point is excluded)
*** Random

Table 9.3: The comparison of the GA with simulated annealing and local search.

that the GA-'E' improves the results presented previously in [314]. GA-'D' seems to be weaker at the larger problems. The possible cause of it is the low value of $mmax$ at the mutation.

9.5 Conclusions

We considered the solving of the 3-matching problem by a genetic algorithm. The algorithm can use many different approaches at the same run. The use of the components can be stated as a simple parameter setting. Our GA which incorporates problem-specific crossover and iterative hill-climbing gave excellent results on various problem instances.

Although the results are acceptable, our GA can be enhanced in many aspects. Our present research is focused on the following issues.

First, the HC operation is time-consuming. It should be replaced with smaller hill-climbing steps or moves. One possibility is to allow only a fixed number of improvements at the triplet interchanges. Another possibility is to fix the number of triplet comparisons.

Second, the knowledge-based recombination operators suffer from a kind of deterministic behaviour, which leads to premature convergence in many cases. We should therefore reconsider these operators and add randomness at the child generation process.

Finally, a very important task is to design a method for the adaptive choice of the recombination operators in order to eliminate the problem of choosing the right combination of available methods beforehand.

Acknowledgements

The authors wish to thank Prof. Timo Raita, M.Sc. Harri Hakonen and two anonymous referees for their valuable help.

9.6 Bibliography

See chapter bibliography at the end of this proceedings.