Ann Oper Res
DOI 10.1007/s10479-015-1848-y

Experimental analysis of crossover and mutation operators on the quadratic assignment problem

Zakir Hussain Ahmed[1]

Abstract In genetic algorithms crossover is the most important operator where pair of chromosomes and crossover site along their common length are selected randomly. Then the information after the crossover site of the parent chromosomes is swapped. On the other hand, mutation operator randomly alters some genes of a chromosome, and thus diversifies the search space. We consider three crossover and ten mutation operators for the genetic algorithms which are then compared for the quadratic assignment problem on some benchmark QAPLIB instances. The experimental study shows the effectiveness of the sequential constructive crossover and the adaptive mutation operators for the problem.

Keywords Quadratic assignment problem · NP-hard · Genetic algorithm · Sequential constructive crossover · Adaptive mutation

1 Introduction

The quadratic assignment problem (QAP) is a NP-hard combinatorial optimization problem (Sahni and Gonzales 1976). It is one of the most difficult combinatorial optimization problems, which was first introduced by Koopmans and Beckmann (1957) as a mathematical model related to economic activities. The problem can be defined as follow: There are a set of n facilities and a set of n locations. For each pair of facilities, a flow matrix, $F = [f_{ij}]$, is specified and for each pair of locations, a distance matrix, $D = [d_{kl}]$, is specified. Also, let $a = \{a(1), a(2), \ldots, a(n)\}$ be an assignment, where $a(i)$ represents the location of the facility i. The problem is to assign to each location exactly one facility so as to minimize the cost (objective function)

✉ Zakir Hussain Ahmed
zhahmed@ccis.imamu.edu.sa; zhahmed@gmail.com

1 Department of Computer Science, College of Computer and Information Sciences, Al Imam Mohammad Ibn Saud Islamic University (IMSIU), P.O. Box No. 5701, Riyadh 11432, Kingdom of Saudi Arabia

Published online: 02 April 2015

$$Z_a = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{a(i)a(j)} \tag{1}$$

The QAP has several practical applications that include backboard wiring (Steinberg 1961), scheduling problems (Geoffrion and Graves 1976), typewriter keyboards and control panels (Pollatschek et al. 1976), hospital planning (Elshafei 1977), archeology (Krarup and Pruzan 1978), economic problems (Heffley 1980), statistical analysis (Hubert 1987), forest parks (Bos 1993), analysis of reaction chemistry (Forsberg et al. 1994), numerical analysis (Brusco and Stahl 2000), placement of electronic components (Miranda et al. 2005; Duman and Ilhan 2007), etc.

Since there are n! possible assignments, so complete enumeration method is impossible for $n > 14$, even if both flow $F = [f_{ij}]$s and distance $D = [d_{kl}]$ matrices are symmetric. Due to the practical importance and complexity of the QAP, it has been drawing attention of many researchers, who developed several exact and heuristic algorithms for solving the problem. Exact algorithms for the QAP include branch-and-bound (Hahn et al. 2001), branch-and-cut (Erdoğan and Tansel 2007), lexisearch (Ahmed 2013a, 2014c), mixed integer linear programming (Zhang et al. 2013). In general, instances of size $n > 30$ cannot be solved optimally by an exact algorithm in reasonable time (Drezner et al. 2005).

Since there are many real-life instances of size more than 30, hence to solve those instances one has to go for heuristics. Heuristics are the methods which do not guarantee the optimality of the solution, but give near exact optimal solution very quickly. Heuristic methods for the QAP as well as combinatorial optimization problems are categorized into: constructive methods (Gilmore 1962), enumerative methods (Burkard and Bonniger 1983), and improvement methods (Mills et al. 2003). The most recent heuristic methods that can be adapted to a wide range of combinatorial optimization problems are called metaheuristics. Examples of such methods for the QAP are guided local search (Mills et al. 2003), greedy algorithms (Oliveira et al. 2004), tabu search (Paul 2011), genetic algorithms (Ahuja et al. 2000), simulated annealing (Wilhelm and Ward 1987), ant colony optimization (Acan 2005), etc. Amongst the heuristics genetic algorithms (GAs) are found to be the best algorithms for the QAP as well as other combinatorial optimization problems.

Genetic algorithms (GAs) are very powerful and robust heuristic approaches for solving large-scale combinatorial optimization problems. They are based essentially on imitating the process of evolution (Goldberg 1989). Each feasible solution of a problem can be treated as a chromosome (or an individual) whose fitness value is measured by its objective function value. A simple GA starts by randomly generating a set of chromosomes, which is called an initial population of chromosomes, also referred as gene pool, and then applies (possibly three) operators to create new, and hopefully, better populations as successive generations. The first operator is selection which probabilistically copies and discards some of the chromosomes of the present generation to the next generation. Crossover is the second operator where randomly selected pairs of chromosomes are mated to create new chromosomes. The third operator is mutation, which randomly alters some position values (genes) of a chromosome. Crossover is the most important operator in the GA search that converges the search space, whereas mutation diversifies the search space. So, the probability of applying mutation is set very low, whereas the probability of crossover is set very high. A simple GA repeatedly applies these operators until either the population converges or until reaches the maximum number of generations (iterations).

In this paper, we consider three crossover and ten mutation operators for the simple GAs to the QAP. A comparative study among the crossover and mutation operators has been carried out on some benchmark QAPLIB instances (Burkard et al. 1997). Experimental results show that the sequential constructive crossover (Ahmed 2014d) is the best crossover operator and adaptive mutation (Ahmed 2014e) is the best mutation operator. It is to be noted that the aim and objective of this study is to investigate the efficiency of the crossover and mutation operators, not to find the optimal solution to the QAP. However, a comparative study between our simple GA and migrating birds optimization of Duman et al. (2012) is reported that shows the effectiveness of our algorithm.

This paper is organized as follows: Section 2 presents a brief review on different crossover and mutation operators for the QAP. Section 3 discusses three different crossover operators, whereas Sect. 4 discusses ten different mutation operators for our simple genetic algorithms. Section 5 reports computational experiments for simple genetic algorithms using different crossover and mutation operators. Finally, Sect. 6 presents concluding remarks and future work.

2 Crossover and mutation operators for the QAP- a review

The GA search of the solution space is done by creating new chromosomes from old ones. As mentioned above, crossover is the most important operator in GAs. In crossover operator, a pair of chromosomes is selected randomly from the mating pool. Then a point, called crossover site, along their common length is randomly selected, and the information after the crossover site of the two parent chromosomes are swapped, thus creating two new children. However, this basic crossover method does not support for the QAP. Most of crossover operators used for the QAP are the modification of the crossover operators for the travelling salesman problem (TSP).

Goldberg and Lingle (1985) defined an operator called partially mapped crossover (PMX) for the TSP that used two crossover points. PMX has been proved to be highly effective for the TSP; however the straightforward PMX procedure does not work well for the QAP. For this reason, Migkikh et al. (1996) proposed a modification of PMX based on using a number of random mapping points - instead of one mapping segment. This crossover was referred to as a uniform PMX (UMPX). The ordered crossover (OX) operator is developed by Davis (1985) for the TSP that builds offspring by choosing a subsequence of a tour from one parent and preserving the relative order of nodes from the other parent. This OX operator has been applied to the QAP by Vázquez and Whitley (2000). One point crossover (OPX) operators (Goldberg 1989) are classical methods which were widely used in early versions of GAs. Lim et al. (2000) proposed a variation of the OPX for the QAP. Another crossover operator, named cycle crossover (CX) operator was proposed for the TSP by Oliver et al. (1987), where offspring are built in such a way that each node (and its position) comes from one of the parents. Merz and Freisleben (2000) applied this crossover operator for the GA to the QAP. The uniform like crossover (ULX) was proposed by Tate and Smith (1995). Misevicius (2004) slightly improved this crossover operator, renamed as optimized crossover (OX) and used in the improved hybrid GA. Misevicius and Kilda (2005) proposed many modification of this crossover and called them as randomized ULX (RULX), modified ULX [or block crossover (BX)], and extended ULX [or repair crossover (RX)]. Ahuja et al. (2000) developed a swap path crossover (SPX) for the QAP. It was proposed to perform the swap for which the corresponding solution has lower cost than the original one. Drezner (2003) proposed another crossover operator, named cohesive crossover (COHX), for the QAP. Misevicius and Rubliauskas (2005) proposed a multiple parent crossover (MPX) operator that creates an offspring by choosing information from many parents. Misevicius and Kilda (2005) made a comparative study among ten different crossover operators for the QAP and found that MPX is better than SPX which is better than OPX which is then better than COHX. Ahmed (2014d) developed a crossover operator, named sequential constructive crossover (SCX), for the QAP and compared with SPX and OPX, and found that SCX is the best.

Mutation operator modifies the value of each gene of a chromosome with some probability. The job of mutation operator is to restore the lost or unexplored genetic material into the population to stop the premature convergence of the GA to suboptimal solutions. The mutation operator selects a position randomly in a chromosome and changes the value of corresponding gene, and hence, modifies the information. Since the weaker members of every generation are rejected; some genetic features could be vanished forever. By allowing random changes in the chromosomes, GAs guarantee that unexplored search space is reached, which selection and crossover couldn't fully guarantee. This is how mutation guarantees that no important features are prematurely vanished, and hence, maintains the diversity of population.

The basic mutation operator does not work for the QAP. In general, mutations used for the TSP are also used for the QAP. Various mutation operators are proposed for the TSP. Some of them are discussed here. Fogel (1988) proposed insertion mutation that selects a gene randomly and inserts it in a random position of a chromosome. Banzhaf (1990) proposed exchange mutation that selects two positions randomly and swaps the genes on these positions. This mutation is basically the 2-opt search. Inversion mutation (Fogel 1990) selects two positions randomly within a chromosome and then inverts the subchromosomes between these two positions. Michalewicz (1992) proposed displacement mutation that selects randomly a subchromosome and inserts it in a random position. This is basically a special case of the displacement mutation in which the subchromosome contains only one gene. Heuristic mutation was proposed by Gen and Cheng (1997) that designed with neighborhood method in order to produce an improved offspring. Greedy swap mutation (Louis and Tang 1999) is similar to the heuristic mutation that selects always better results and therefore it gives near solution very quickly. Though it reaches a near solution quickly, but it is very difficult to come out from local solutions which are far from optimal. It is very common that selection of mutation operator and mutation probability takes long time in order to optimize the performance of GA. Hence, Serpell and Smith (2010) proposed self-adaptation of mutation operators for the 0/1 knap-sack problem and found that the self-adaptation of mutation probability takes precedence over the self-adaptation of mutation operator. Deep and Mebrahtu (2011) proposed two combined mutation operators—inverted exchange and inverted displacement for increasing the performance of GA. All the above mutation operators can be applied to the QAP. Ahmed (2014e) proposed an adaptive mutation for the QAP, and found to be better than the exchange mutation.

3 Crossover operators for our GAs

For the QAP, solutions are naturally represented by permutation of positive integers of length equal to the problem size (total number of locations or facilities), n, where each gene of a chromosome can take any number in {1, 2, 3,…, n}. The jth number in the permutation denotes the location of the facility j. For a 7-location instance, (7, 5, 2, 3, 4, 6, 1) may be chromosome, where 1st facility is located at 7th site, 2nd facility is at 5th site, and so on. Accordingly, we generate randomly a set of chromosomes which is considered as an initial population of prefixed population size. We define the 'fitness function' as multiplicative inverse of the objective function. As a selection method, we have considered the stochastic remainder selection method (Deb 1995) to create a mating pool for our GAs. As mentioned above, crossover operator is the most important process in GAs. We consider the following three crossover operators for the comparative study.

3.1 Sequential constructive crossover operator

The sequential constructive crossover (SCX) was initially developed for the TSP (Ahmed 2010), which was then applied successfully to the TSP with several variations (Ahmed 2011, 2013b, c, 2014a, b). The SCX is applied to the QAP (Ahmed 2014d) and found to be better than one point crossover (Lim et al. 2000) and swap path crossover (Ahuja et al. 2000). The SCX for the QAP is as follows.

Step 1: Start from the location (suppose, p) of the first facility of any randomly chosen parent.
Step 2: Sequentially search both of the parent chromosomes and consider the first 'legitimate location' (the location that is not yet assigned) appeared after 'location p' in each parent. If no any 'legitimate location', after 'location p', is available in any of the parent, search sequentially from the beginning of the parent and consider the first 'legitimate location', and go to Step 3.
Step 3: Suppose 'location α' and 'location β' are found in 1st and 2nd parent respectively, then for selecting the next location go to Step 4.
Step 4: Compute the cost of one incomplete offspring chromosome by incorporating 'location α' as the next location (suppose, c_α). Similarly, compute the cost of other incomplete offspring by incorporating 'location β' as the next location (suppose, c_β). Then go to Step 5. Suppose $(\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_{k-1})$ be a partially constructed offspring and 'location δ' is selected for concatenation, then the cost (value) of assigning this location for the facility k is calculated as follows:

$$c_\delta = \sum\nolimits_{i=1}^{k-1} \left(f_{ik} d_{\alpha_i \delta} + f_{ki} d_{\delta \alpha_i} \right) \tag{2}$$

Step 5: If $c_\alpha \leq c_\beta$, then select 'location α', otherwise, 'location β' as the next location to be assigned for the next facility and concatenate it to the partially constructed offspring chromosome. If the offspring is a complete chromosome, go to Step 6; otherwise, rename the present location as 'location p' and go to Step 2.
Step 6: Evaluate the first parent and the offspring chromosomes. If value of the offspring is less than the value of the parent, replace the first parent by the offspring, otherwise skip it.

Let us illustrate the SCX through the example given as flow and distance matrices in Tables 1 and 2 respectively. Let a pair of selected chromosomes be P$_1$: (5, 2, 3, 4, 1, 7, 6) and P$_2$: (3, 5, 7, 4, 1, 2, 6) with values 1423 and 1347 respectively. Suppose 'location 5' is selected randomly as the 1st gene. The step-by-step illustration for obtaining an offspring is shown in Table 3.

Thus the complete offspring chromosome will be (5, 7, 6, 3, 4, 1, 2) with value (40 + 78 + 240 + 343 + 286 + 268 =) 1255 which is less than the value of both parent chromosomes.

3.2 One point crossover operator

The one point crossover (OPX) was proposed by Lim et al. (2000). The idea of OPX is quite simple. A crossing point (site) is chosen randomly between 1 and n − 1 in one of the parents. As a result, an offspring chromosome is obtained, containing information partially determined by each of parent chromosomes. Again, the care of preserving the solution feasibility should

Table 1 The flow matrix **F**

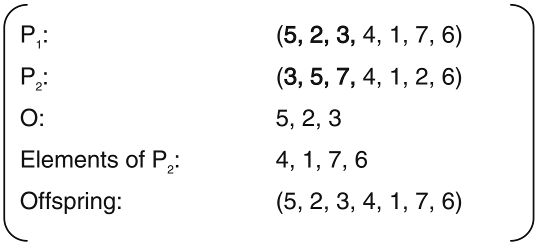
Facility	1	2	3	4	5	6	7
1	X	5	3	7	9	3	9
2	5	X	7	8	3	2	3
3	3	7	X	9	3	5	3
4	7	8	9	X	8	4	1
5	9	2	2	8	X	8	8
6	3	2	4	4	9	X	4
7	8	4	1	1	8	4	X

Table 2 The distance matrix **D**

Location	1	2	3	4	5	6	7
1	X	7	4	6	8	8	8
2	7	X	8	2	6	5	6
3	4	8	X	10	4	4	7
4	6	2	10	X	6	6	9
5	8	6	4	6	X	6	4
6	8	5	4	6	6	X	3
7	8	6	7	9	4	3	X

Table 3 Step-by-step illustration for obtaining an offspring

Location p	Legitimate location after p in the parents	Corresponding costs	Selected location	Offspring
5	{2, 7}	{60, **40**}	7	(5, 7)
7	{6, 4}	{**78**, 162}	6	(5, 7, 6)
6	{2, 3}	{247, **240**}	3	(5, 7, 6, 3)
3	{4, 4}	{**343**, 343}	4	(5, 7, 6, 3, 4)
4	{1, 1}	{**286**, 286}	1	(5, 7, 6, 3, 4, 1)
1	{2, 2}	{**268**, 268}	2	(5, 7, 6, 3, 4, 1, 2)



P_1:	(**5, 2, 3,** 4, 1, 7, 6)
P_2:	(**3, 5, 7,** 4, 1, 2, 6)
O:	5, 2, 3
Elements of P_2:	4, 1, 7, 6
Offspring:	(5, 2, 3, 4, 1, 7, 6)

Fig. 1 Example of OPX operation

be taken. Let us consider the same pair of parents P_1 and P_2 in Sect. 3.1, and suppose, three is a crossing site. Then there is no improvement found by the operation for the given example. The operation is shown in Fig. 1.

3.3 Swap path crossover operator

The swap path crossover (SPX) was proposed by Ahuja et al. (2000), which is described as follows. Let P_1, P_2 be a pair of parents. In SPX, one starts at the first (or some random) gene, and the parents are examined from left to right until all the genes have been considered. If the alleles at the position being looked at are the same, one moves to the next position; otherwise, one performs a swap (interchange) of two alleles in P_1 or in P_2 so that the alleles at the current position become alike. (For example, if the current gene is i, and $a = P_1(i)$, $b = P_2(i)$, then, after a swap, either $P_1(i)$ becomes (b), or $P_2(i)$ becomes (a). Ahuja et al. (2000) proposed to perform the swap for which the corresponding solution has a lower cost (objective function value). The elements in the two resulting solutions are then considered, starting at the next position, and so on. The best solution obtained during this process (the fittest child) serves as an offspring. A part of SPX operator is illustrated in Fig. 2. The best offspring chromosome will be (3, 5, 2, 4, 1, 7, 6) with value 1301 which is less than the value of both parent chromosomes. That is, an improvement is found by the operation for the given example. However, for the given example, we see that our SCX gives better offspring than OPX and SPX.

4 Mutation operators for our GA

Mutation is a process of increasing diversity in the population by introducing random variations in the population. It randomly selects a gene in a chromosome and changes the corresponding gene, thereby modifies the information.

P_1:	(5, 2, 3, 4, 1, 7, 6)
P_2:	(3, 5, 7, 4, 1, 2, 6)
O_1:	(3, 2, 5, 4, 1, 7, 6) with value 1312
O_2:	(5, 3, 7, 4, 1, 2, 6) with value 1447
Better Offspring:	(3, 2, 5, 4, 1, 7, 6) with value 1312
New P_1:	(3, 2, 5, 4, 1, 7, 6)
New P_2:	(3, 5, 7, 4, 1, 2, 6)
O_1:	(3, 5, 2, 4, 1, 7, 6) with value 1301
O_2:	(3, 2, 7, 4, 1, 5, 6) with value 1354
Better Offspring:	(3, 5, 2, 4, 1, 7, 6) with value 1301
New P_1:	(3, 5, 2, 4, 1, 7, 6)
New P_2:	(3, 5, 7, 4, 1, 2, 6)
O_1:	(3, 5, 7, 4, 1, 2, 6) with value 1347
O_2:	(3, 5, 2, 4, 1, 7, 6) with value 1301
… ……	……… …… …… …… …… ………
Best Offspring:	(3, 5, 2, 4, 1, 7, 6) with value 1301.

Fig. 2 Example of SPX operation

4.1 Existing mutation operators

We explain eight existing mutation operators. These operators are—exchange mutation, displacement mutation, insertion mutation, inversion mutation, inverted exchange mutation, inverted displacement mutation, repaired exchange mutation, and adaptive mutation. In addition to these operators two proposed mutation operators will be described. We discuss all the mutation operators through the chromosome, P: (5, 2, 3, 4, 1, 7, 6).

4.1.1 Exchange mutation

Exchange mutation selects two positions randomly and swaps the genes on these positions. For example, if positions 1 and 6 are selected randomly, facility 5 and facility 7 are interchanged their positions, then the mutated chromosome (O) will be

$$\text{O}: (\mathbf{7}, 2, 3, 4, 1, \mathbf{5}, 6)$$

4.1.2 Displacement mutation

Displacement mutation randomly selects a subchromosome and inserts it at a random position outside the subchromosome. For example, if the randomly selected subchromosome is (2, 3, 4) and the randomly selected insertion position is between 7 and 6, then the mutated chromosome (O) will be

$$\text{O}: (5, 1, 7, \mathbf{2}, \mathbf{3}, \mathbf{4}, 6)$$

4.1.3 Insertion mutation

Insertion mutation selects a gene randomly and inserts it at random position in a chromosome. It can be viewed as a particular case of displacement mutation where the subchromosome contains only one gene. For example, if the randomly selected gene is 3 and the random position is between 1 and 7, then the mutated chromosome (O) will be

$$\text{O}: (5, 2, 4, 1, 7, \mathbf{3}, 6)$$

4.1.4 Inversion mutation

Inversion mutation selects randomly two positions within a chromosome and then inverts the genes in the subchromosome between these two positions. For example, if a subchromosome (3, 4, 1) is selected randomly using 3rd and 5th positions, then the mutated chromosome (O) will be

$$\text{O}: (5, 2, \mathbf{1}, \mathbf{4}, \mathbf{3}, 7, 6)$$

4.1.5 Inverted exchange mutation

In this method, first select two positions randomly within a chromosome and then invert the genes in the subchromosome between these two positions. Next, select randomly one gene from the inverted subchromosome and another gene from outside the inverted subchromosome, and then swap them.

For example, if a subchromosome (3, 4, 1) is selected randomly between 3rd and 5th positions, then after inverting the subchromosome, the parent chromosome will become (5, 2, **1, 4, 3,** 7, 6). Next, if gene **4** from the inverted subchromosome and gene **5** from outside the inverted subchromosome, and then after swapping them muted chromosome (O) will be (**4**, 2, 1, **5**, 3, 7, 6).

4.1.6 Inverted displacement mutation

First, select two positions randomly within a chromosome and then invert the genes in the subchromosome between these two positions. Next, the inverted subchromosome is inserted at a random position outside the subchromosome.

For example, select 2nd and 5th positions randomly within the chromosome. Then after inverting the subchromosome (2, 3, 4, 1), the chromosome becomes (5, **1, 4, 3, 2,** 7, 6). Next, inserting the inverted subchromosome randomly at 7th position, the muted chromosome (O) will be

$$\mathrm{O}: (5, 7, \mathbf{1}, \mathbf{4}, \mathbf{3}, \mathbf{2}, 6)$$

4.1.7 Repaired exchange mutation

Repaired exchange mutation selects two positions randomly and swaps the genes on these positions, if the muted chromosome is better than the parent chromosome.

4.1.8 Adaptive mutation

In this method, data are collected from all chromosomes in the current population to detect a pattern among them. If the mutation takes place, then chromosomes that do not resemble the pattern will be muted. In fact, this is an intelligent exchange mutation. The algorithm is as follows (Ahmed 2014e; Ahmed et al. 2014):

Step 1: Consider all chromosomes in the current population.
Step 2: Create a one-dimensional array of size n (size of the problem), suppose, A, by storing a location (gene) that appears minimum number of times in the current position of all chromosomes.
Step 3: If mutation is allowed, select randomly two genes such that they are not same in the corresponding positions of the array, A, and swap them.

For example, suppose the one dimensional array is A = [2, 3, 6, 2, 5, 7, 1] and 3rd and 6th positions are selected randomly. The 3rd position's gene '3' does not match array element in the position, but 6th position's gene '7' matches the corresponding array element. So, we select another position and suppose, the 7th is selected randomly that allows the exchange. Hence, the mutated chromosome (O) will be

$$\mathrm{O}: (5, 2, \mathbf{6}, 4, 1, 7, \mathbf{3})$$

4.2 The proposed mutation operators

The proposed mutation operators are 3-exchange mutation and gene-exchange mutation.

4.2.1 3-exchange mutation

3-exchange mutation selects three different positions at random, say, r_1, r_2 and r_3, and swaps the genes on these positions as follows: $P(r_1) \leftrightarrow P(r_2)$ and then $P(r_2) \leftrightarrow P(r_3)$. For example, if positions 1, 4 and 6 are selected randomly, then facility 5 and facility 4 are interchanged their positions that leads to the chromosome (4, 2, 3, 5, 1, 7, 6), and then facility 5 and facility 7 are interchanged their positions that leads to the mutated chromosome (O) as

$$O: (\mathbf{4}, 2, 3, \mathbf{7}, 1, \mathbf{5}, 6)$$

4.2.2 Gene-exchange mutation

Gene-exchange mutation selects two genes randomly and swaps them. For example, if genes (facilities) 1 and 6 are selected randomly, then the mutated chromosome (O) will be

$$O: (5, 2, 3, 4, \mathbf{6}, 7, \mathbf{1})$$

Our simple GA may be summarized as follows.

```
Genetic Algorithm ( )
{       Initialize population;
        Evaluate the population;
        Generation = 0;
        While termination condition is not satisfied
        {       Generation = Generation + 1;
                Select good chromosomes by selection procedure;
                Perform crossover with crossover probability ($P_c$);
                Perform mutation with mutation probability ($P_m$);
                Evaluate the population;
        }
}
```

5 Computational experiments

As an experimental basis, simple GAs using different crossover and mutation operators have been encoded in Visual C++ and run on a PC with Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz and 8.00GB RAM under MS Windows 7. In the experiments, we used some benchmark QAPLIB instances (Burkard et al. 1997). In particular, random and real-life instances proposed by Taillard (1995) are used. In QAPLIB, random instances are denoted by tai20a, tai25a, tai30a, tai35a, tai40a, tai50a, tai60a, tai80a, and tai100a (in short, tai*a), where the corresponding numeral in the instance name indicates the size of the problem. These instances belong to the class of regular, unstructured instances that are generated using uniform distribution. Real-life instances are denoted by tai20b, tai25b, tai30b, tai35b, tai40b, tai50b, tai60b, tai80b, and tai100b (in short, tai*b). They are generated randomly, but not uniformly, so that the values of the flow and distance matrices resemble a distribution from real-world problems. The experiments were performed hundred times for each instance. The solution quality is measured by the percentage of excess [Excess (%)] of solution value

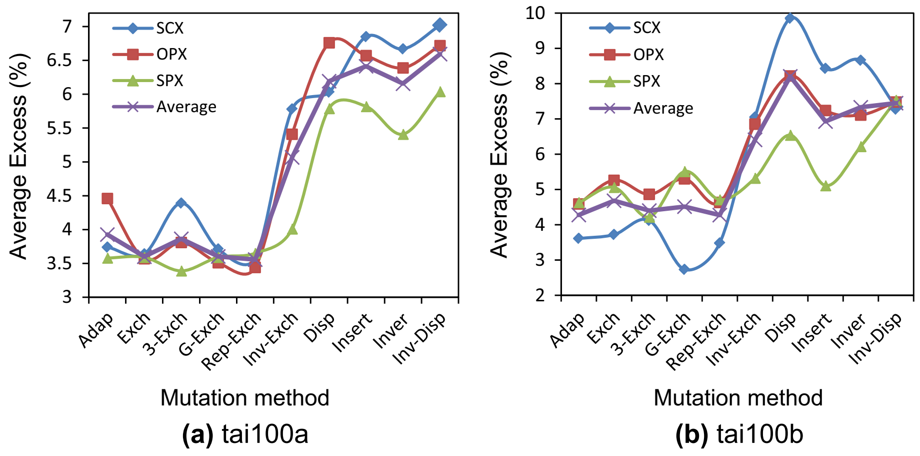


Fig. 3 Comparative study of different mutation operators on tai100a and tai100b

(SV) obtained from the best known solution value (BKV) reported in QAPLIB, as given by the formula

$$Excess(\%) = 100 \times (SV - BKV)/BKV \qquad (3)$$

We report percentage of excess of best solution value and average solution value over the best known solution value (BKV) of 100 runs. To avoid bias towards any particular algorithm, we use same randomly generated population for 1st run of all algorithms, same population for 2nd run of all algorithms, and so on for all 100 runs. We also report average time of convergence (in seconds) by the algorithms. It is to be noted that GAs are controlled by some parameters, namely, population size, crossover probability, mutation probability, and termination criterion. To have a clear picture of crossover operators, we set 1.0 (i.e., 100 %) as crossover probability (P_c), and approximately same computational time (explicitly mentioned in Table 5) as a termination criterion. Also, after having extensive testing, we set 30 as population size (P_s), and 0.1 (i.e., 10 %) as mutation probability (P_m). However, it is very difficult to set common parameters for all instances.

First of all, we want to see the performance of the mutation operators on the instances tai100a and tai100b. For measuring the performance, average excess (%) is considered. Due to the similarity of results from different QAPLIB instances, we present results of tai100a and tai100b only. Figure 3a, b shows comparative study for the simple GAs using three crossover operators and ten different mutation operators on tai100a and tai100b respectively. From the figure it is seen that only the first five mutation operators are performing well, others not. We also conducted analysis of variance (ANOVA) tests to confirm our observation about the performance of these ten mutation methods for tai100a and tai100b. We consider the critical value for this analysis is as 0.05. After performing ANOVA test, we are 95 % confident that the average excess (%) is not statistically equal for all mutation methods. However, an ANOVA test just tests that at least one average is different; not showing us which one is different. Hence, we have conducted the two-tailed student *t* test with 5 % significant level between adaptive mutation and other mutation operators for both tai100a and tai100b together.

Table 4 summarizes the mean and variance of average excess (%) of the instances tai100a and tai100b using three crossover and ten mutation operators, the two-tail *p* value of the *t* test of the differences between the average excesses (%) of adaptive mutation compared with

Table 4 The result of *t* test between adaptive mutation and other mutation methods

	Adap	Exch	3-Exch	G-Exch	Rep-Exch	Inv-Exch	Disp	Insert	Inver	Inv-Disp
Mean	4.05	4.09	4.08	4.01	3.87	5.69	7.15	6.62	6.69	6.97
Variance	0.25	0.62	0.25	1.20	0.34	1.23	2.40	1.31	1.18	0.30
p value	–	0.868	0.894	0.895	0.327	0.018	0.006	0.007	0.004	0.000
t-statistic	–	0.175	0.140	0.138	1.086	3.479	4.592	4.438	5.062	13.423
Critical *t* value		2.571								

the corresponding average excess (%) of other nine mutation operators, and the t-statistics. Since *p* values of the differences between adaptive mutation and other four mutations, that is, exchange mutation, 3-exchange mutation, gene-exchange and repaired-exchange, are greater than the significant level, 0.05, hence these five mutations are statistically equal. On the other hand, since *p* values of the differences between adaptive mutation and other remaining five mutations are less than the significant level, hence these five mutations are statistically different from the first five mutations. Also, by looking at the t-statistic values and comparing with the critical *t* value, we come to the same conclusion. So, we confirm that the first five mutation operators are statistically different from (and better than) the last five mutation methods, and the first five mutation methods are statistically equal. Therefore, for the further study we consider only the first five mutation methods and compare them on other QAPLIB instances.

Table 5 shows comparative study among simple GAs using three crossover and five mutation operators on ten QAPLIB instances. In first column, we report instance name, its best known solution value reported in QAPLIB website and prefixed computational time in seconds within brackets as termination criterion. Second column reports name of the crossover operator and last column reports grand average of the excess (%) using a particular crossover operator and all mutation operators. The remaining columns report the best excess (%) and average excess (%) using different mutation operators as mentioned therein.

Irrespective of any mutation operator, in Table 5, by looking at the grand average of excess (%) (in boldface), SCX is found to be best for tai40a, tai40b, tai50a, tai50b, tai60a, tai60b, tai80b, and tai100b; and SPX is found to be best for remaining two instances—tai80a and tai100a. The grand average excess (%) that is shown in Table 5 is also depicted in Fig. 4. Hence, SCX is found to be best one, and OPX is found to be the worst one. Similarly, irrespective of any crossover operator, by looking at the partial average of average excess (%) (in boldface), adaptive mutation is found to be the best for tai40b, tai50a, tai50b, tai60b, and tai80b; exchange mutation is best for tai80a, and tai100a; 3-exchange is best for tai40a, and tai60a; gene-exchange is best for tai100a; and repaired-exchange is best for tai100b. Overall, by looking at the partial averages of best excesses (%) and average excesses (%), i.e., the last row, it is very clear that adaptive mutation is the best one, then 3-exchange, then gene-exchange, then exchange, and lastly, repaired-exchange mutation.

Regarding the combination crossover and mutation operators, by looking at the average excess (%) (mark by an asterisk), combination of SCX and adaptive mutation is found to be best for tai40b, tai50b and tai80b; combination of SCX and exchange mutation is best for tai60a; combination of SCX and gene-exchange mutation is best for tai100b. Next, combination of SPX and adaptive mutation is found to be best for tai40a and tai80a; combination of SPX and 3-exchange mutation is best for tai50a and tai100a. Further, combination of OPX and adaptive mutation is found to be best for tai60b only. Hence, combination of SCX and

Table 5 Comparison of the crossover and mutation operators for some QAPLIB instances

Instance	Cross	Adaptive		Exchange		3-Exchange		G-Exchange		Rep-Exchange		Grand Avg. (%)
		Best (%)	Avg. (%)	Best (%)	Avg. (%)	Best (%)	Avg. (%)	Best (%)	Avg. (%)	Best (%)	Avg. (%)	
tai40a	SCX	2.53	3.80	2.78	4.13	2.78	3.73	3.72	4.30	3.58	4.26	**3.56**
3139370	OPX	2.64	4.47	3.19	4.20	2.62	3.92	3.04	4.28	3.32	4.63	3.63
(20 Sec)	SPX	2.24	3.63*	3.16	4.80	2.45	3.81	3.03	4.51	3.49	4.57	3.57
Partial Avg.		2.47	3.97	3.04	4.38	2.62	**3.82**	3.26	4.36	3.46	4.49	
tai40b	SCX	0.38	3.38*	4.23	6.43	0.95	5.26	1.00	5.46	3.58	6.55	**3.72**
637250948	OPX	0.11	4.33	2.59	8.35	0.40	7.82	0.00	6.77	1.85	8.09	4.03
(20 Sec)	SPX	1.68	5.66	5.03	8.78	0.02	7.55	3.14	8.81	4.48	8.61	5.38
Partial Avg.		0.72	**4.46**	3.95	7.85	0.46	6.88	1.38	7.01	3.30	7.75	
tai50a	SCX	3.06	3.92	3.61	4.28	3.59	4.07	3.39	4.02	3.57	4.20	**3.77**
4938796	OPX	3.65	4.81	2.77	4.05	2.96	4.01	3.35	4.35	3.40	4.68	3.80
(30 Sec)	SPX	3.22	4.14	3.70	4.69	2.43	3.74*	3.76	4.54	3.35	4.47	3.80
Partial Avg.		3.31	**4.29**	3.36	4.34	2.99	3.94	3.50	4.30	3.44	4.45	
tai50b	SCX	0.84	2.54*	1.38	3.86	1.91	4.78	1.59	4.47	1.05	3.72	**2.61**
458821517	OPX	1.04	3.90	0.27	6.35	2.43	6.94	1.08	6.44	3.06	6.61	3.81
(30 Sec)	SPX	0.10	4.13	0.46	5.87	0.76	5.81	1.46	5.04	0.31	4.78	2.87
Partial Avg.		0.66	**3.52**	0.70	5.36	1.70	5.84	1.38	5.32	1.47	5.04	
tai60a	SCX	3.37	3.85	3.09	3.62*	3.67	4.21	3.44	4.00	3.56	4.05	**3.69**
7205962	OPX	3.86	4.61	3.36	4.22	3.39	3.88	2.99	4.14	3.37	4.39	3.82
(40 Sec)	SPX	2.75	3.89	3.54	4.37	2.90	2.90	3.20	4.31	3.92	4.59	3.74
Partial Avg.		3.33	4.12	3.33	4.07	3.32	**4.00**	3.21	4.15	3.62	4.34	
tai60b	SCX	1.25	3.57	2.62	4.10	0.74	3.77	1.52	4.14	3.07	5.20	**3.00**
608215054	OPX	0.49	2.98*	0.49	5.28	1.26	7.93	0.60	6.83	0.72	7.31	3.39
(40 Sec)	SPX	1.49	5.18	2.71	6.70	1.97	6.86	1.47	7.06	1.33	6.08	4.09

Table 5 continued

Instance	Cross	Adaptive		Exchange		3-Exchange		G-Exchange		Rep-Exchange		Grand Avg. (%)
		Best (%)	Avg. (%)	Best (%)	Avg. (%)	Best (%)	Avg. (%)	Best (%)	Avg. (%)	Best (%)	Avg. (%)	
Partial Avg.		1.08	**3.91**	1.94	5.36	1.32	6.19	1.20	6.01	1.71	6.20	
tai80a	SCX	3.58	3.81	3.35	3.68	4.07	4.37	3.53	3.78	3.34	3.76	3.73
13499184	OPX	3.97	4.52	3.12	3.71	3.35	3.82	3.16	3.67	3.26	3.86	3.64
(75 Sec)	SPX	3.00	3.60*	2.95	3.75	3.12	3.63	3.10	3.79	3.34	4.05	**3.43**
Partial Avg.		3.52	3.98	3.14	**3.71**	3.51	3.94	3.26	3.75	3.31	3.89	
tai80b	SCX	1.70	3.63*	2.64	3.99	2.77	4.24	3.14	4.29	2.76	4.02	**3.32**
818415043	OPX	1.76	4.34	2.49	5.21	2.82	4.98	3.13	5.01	2.55	5.72	3.80
(75 Sec)	SPX	2.17	4.56	3.06	5.11	2.42	4.97	1.66	5.27	1.32	5.24	3.58
Partial Avg.		1.88	**4.18**	2.73	4.77	2.67	4.73	2.64	4.86	2.21	4.99	
tai100a	SCX	3.49	3.64	3.38	3.54	4.13	4.29	3.22	3.61	3.17	3.48	3.60
21052466	OPX	3.87	4.36	2.89	3.47	3.15	3.71	2.98	3.41	2.72	3.34	3.39
(120 Sec)	SPX	2.89	3.48	2.92	3.49	2.70	3.29*	2.87	3.49	3.11	3.55	**3.18**
Partial Avg.		3.42	3.83	3.06	**3.50**	3.33	3.76	3.02	**3.50**	3.00	3.46	
tai100b	SCX	1.65	3.51	1.57	3.62	1.70	4.02	1.26	2.63*	1.79	3.38	**2.51**
1185996137	OPX	2.04	4.49	1.45	5.16	1.14	4.76	2.95	5.20	1.60	4.53	3.33
(120 Sec)	SPX	1.68	4.53	1.25	4.96	1.45	4.12	1.27	5.41	1.47	4.61	3.08
Partial Avg.		1.79	4.18	1.42	4.58	1.43	4.30	1.83	4.41	1.62	**4.17**	
Average		**2.22**	**4.04**	2.67	4.79	2.33	4.74	2.47	4.76	2.71	4.88	

Table 6 A comparative study of MBO and SGA for some QAPLIB instances

Instance	BKV	MBO		SGA				
		BSV	Excess (%)	BSV	Excess (%)	P_s	P_m	Time
esc32e	2	2	0.00	2	0.00	All	All	0.22
esc32f	2	2	0.00	2	0.00	All	All	0.23
esc32g	6	6	0.00	6	0.00	All	All	0.18
esc32h	438	438	0.00	438	0.00	Many	Many	5.12
esc64a	116	116	0.00	116	0.00	Many	Many	2.29
tai64c	1,855,928	1,855,928	0.00	1,855,928	0.00	Many	Many	20.3
lipa40b	476,581	476,581	0.00	476,581	0.00	90	0.09	19.47
sko49	23,386	23,420	**0.15**	23,502	0.48	90	0.09	29.59
wil50	48,816	48,834	**0.04**	48,920	0.20	100	0.15	92.19
tai60b	608,215,054	608,231,607	**0.00**	610,054,421	0.30	70	0.14	98.11
lipa70a	169,755	170,817	0.63	170,119	**0.21**	40	0.17	50.58
lipa80a	253,195	254,729	0.61	253,979	**0.31**	50	0.18	174.77
Average			0.119		0.125			41.09

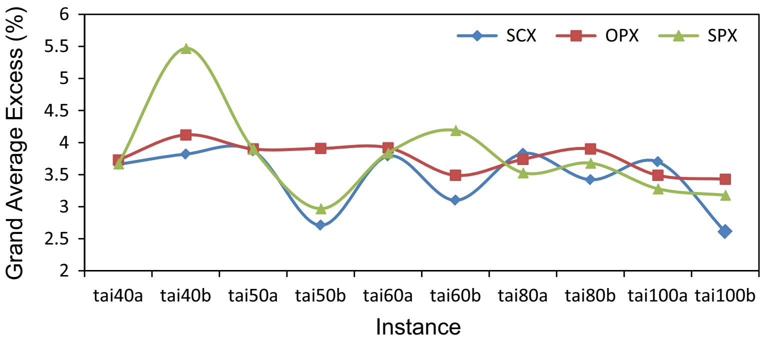


Fig. 4 Comparative study of three different crossover operators

adaptive mutation is the best one. This combination is considered as our simple GA (SGA) for further study. Now, we present a comparative study between the results of our SGA and the results reported in Duman et al. (2012) using migrating birds optimization (MBO).

Table 6 shows comparative study between SGA and MBO on twelve QAPLIB instances. As done by Duman et al. (2012), each problem instance is solved with a number of different values for parameters, here, population size (from 10 to 100 with a step size of 10), mutation probability (from 0.01 to 0.20 with a step size of 0.01), and a maximum of 20,000 generations as termination condition. The best solution value (BSV) with its excess (%), the parameter values giving the best solution, and the computational time (in second) when the best solution is seen for the first time, are reported in the table. If all or most of the parameter values give the best solution, the words 'all' or 'many' is used to indicate. It is seen from the table that for the first seven instances, both algorithms obtain same results. For the remaining five instances, MBO is found to be better for sko49, wil50, and tai60b; whereas SGA is better for lipa70a, and lipa80a. Overall, though MBO is found to be better than the SGA, but the difference between their performances is very small, that is, very negligible. However, a more carefully fine tuned set of parameters may obtain better results. Hence, SGA is competing with MBO.

6 Conclusion and future work

We have developed simple genetic algorithms using three different crossover and ten different mutation operators for the quadratic assignment problem (QAP).Then we performed experiments on the benchmark QAPLIB instances. To see the real picture of the different operators, we use same computational time as stopping condition. From our experimental study it can be seen that sequential constructive crossover is the best. Also, it is seen that the adaptive mutation, exchange mutation, 3-exchange mutation and gene-exchange mutation operators are competing. However, with respect to the average solution quality, adaptive mutation is found to be the best one. Finally, it is found that the combination of sequential constructive crossover and adaptive mutation is the best.

In this present study, our aim was only to compare different crossover and mutation operators with respect to solution quality. We do not aim to improve the solution quality, and hence, we do not use any local search technique to improve the solution quality. Though combination of sequential constructive crossover and adaptive mutation is found to be the best, however, it does not obtain exact optimal solution to all of the examined instances. However, a carefully fine tuned set of parameters, that is, mutation probability, population size, etc., may obtain better results. Finally, we compared the performance of our simple genetic algorithm that uses sequential constructive crossover and adaptive mutation, with migrating birds optimization of Duman et al. (2012), and found that our algorithm is competing with that algorithm. Some literatures (Puchinger and Raidl 2005) combine exact algorithms with metaheuristics to solve optimization problems and found very good results. Our future plan is to combine lexisearch algorithm (Ahmed 2013a, 2014c) and genetic algorithm (Ahmed 2014d) to obtain heuristically optimal solution to the QAP.

Acknowledgments The author is very much thankful to the honorable reviewers for their comments and constructive suggestions which helped the author to improve the paper. This research was supported by the NSTIP strategic technologies program number (10) in the Kingdom of Saudi Arabia vide Award No.11-INF1788-08.The author is very much thankful to the NSTIP for its financial and technical supports.

References

Acan, A. (2005). An external partial permutations memory for ant colony optimization. *Lecture Notes in Computer Science*, *3448*, 1–11.

Ahmed, Z. H. (2010). Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator. *International Journal of Biometrics and Bioinformatics*, *3*(6), 96–105.

Ahmed, Z. H. (2011). Multi-parent extension of sequential constructive crossover for the traveling salesman problem. *International Journal of Operational Research*, *11*(3), 331–342.

Ahmed, Z. H. (2013a). A new reformulation and an exact algorithm for the quadratic assignment problem. *Indian Journal of Science and Technology*, *6*(4), 4368–4377.

Ahmed, Z.H. (2013b). A hybrid genetic algorithm for the bottleneck traveling salesman problem. *ACM Transactions on Embedded Computing Systems*, *12*(1), Art No. 9.

Ahmed, Z. H. (2013c). An experimental study of a hybrid genetic algorithm for the maximum travelling salesman problem. *Mathematical Sciences*, *7*(1), 1–7.

Ahmed, Z. H. (2014a). The ordered clustered travelling salesman problem: A hybrid genetic algorithm. *The Scientific World Journal*, *2014*(258207), 13. doi:10.1155/2014/258207.

Ahmed, Z. H. (2014b). Improved genetic algorithms for the traveling salesman problem. *International Journal of Process Management and Benchmarking*, *4*(1), 109–124.

Ahmed, Z. H. (2014c). A data-guided lexisearch algorithm for the quadratic assignment problem. *Indian Journal of Science and Technology*, *7*(4), 480–490.

Ahmed, Z. H. (2014d). A simple genetic algorithm using sequential constructive crossover for the quadratic assignment problem. *Journal of Scientific and Industrial Research*, *73*(12), 763–766.

Ahmed, Z.H. (2014e). An improved genetic algorithm using adaptive mutation operator for the quadratic assignment problem. In: *Proceedings of 37th international conference on telecommunications and signal processing 2014 (TSP 2014)*, (pp. 616–620). Berlin, Germany.

Ahmed, Z.H., Bennaceur, H., Vulla, M.H., & Altukhaim, F. (2014). A hybrid genetic algorithm for the quadratic assignment problem. In: *Proceedings of second international conference on emerging research in computing, information, communication and applications (ERCICA 2014)*, Vol. *3* (pp. 916–922). Bangalore, India.

Ahuja, R., Orlin, J. B., & Tiwari, A. (2000). A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research*, *27*(10), 917–934.

Banzhaf, W. (1990). The molecular traveling salesman. *Biological Cybernetics*, *64*, 7–14.

Bos, J. (1993). A quadratic assignment problem solved by simulated annealing. *Journal of Environmental Management*, *37*(2), 127–145.

Brusco, M. J., & Stahl, S. (2000). Using quadratic assignment methods to generate initial permutations for least-squares unidimensional scaling of symmetric proximity matrices. *Journal of Classification*, *17*(2), 197–223.

Burkard, R. E., & Bonniger, T. (1983). A heuristic for quadratic Boolean programs with applications to quadratic assignment problems. *European Journal of Operation Research*, *13*, 374–386.

Burkard, R.E., Cela, E., Karisch, S.E., & Rendl, F. (1997). QAPLIB - a quadratic assignment problem library. *Journal of Global Optimization*, *10*, 391–403. See also at http://www.seas.upenn.edu/qaplib/.

Davis, L. (1985). Job-shop scheduling with Genetic Algorithms. In: *Proceedings of an international conference on genetic algorithms and their applications*, (pp. 136–140).

Deb, K. (1995). *Optimization for Engineering Design: Algorithms and Examples*. New Delhi, India: Prentice Hall of India Pvt. Ltd.

Deep, K., & Mebrahtu, H. (2011). Combined mutation operators of genetic algorithm for the travelling salesman problem. *International Journal of Combinatorial Optimization Problems and Informatics*, *2*(3), 1–23.

Drezner, Z. (2003). A new genetic algorithm for the quadratic assignment problem. *Informs Journal on Computing*, *15*(3), 320–330.

Drezner, Z., Hahn, P. M., & Taillard, E. D. (2005). Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. *Annals of Operations Research*, *139*, 65–94.

Duman, E., & Ilhan, O. (2007). The quadratic assignment problem in the context of the printed circuit board assembly process. *Computers and Operations Research*, *34*, 163–179.

Duman, E., Uysal, M., & Alkaya, A. F. (2012). Migrating Birds Optimization: A new metaheuristic approach and its performance on quadratic assignment problem. *Information Sciences*, *217*, 65–77.

Elshafei, A. N. (1977). Hospital layout as a quadratic assignment problem. *Operations Research Quarterly*, *28*(1), 167–179.

Erdoğan, G., & Tansel, B. (2007). A branch-and-cut algorithm for the quadratic assignment problems based on linearizations. *Computers and Operations Research*, *34*, 1085–1106.

Fogel, D. (1990). A parallel processing approach to a multiple travelling salesman problem using evolutionary programming. In: *Proceedings of the fourth annual symposium on parallel processing, Fullerton, California* (pp. 318–326).

Fogel, D. B. (1988). An evolutionary approach to the travelling salesman problem. *Biological Cybernetics*, *60*(2), 139–144.

Forsberg, J. H., Delaney, R. M., Zhao, Q., Harakas, G., & Chandran, R. (1994). Analyzing lanthanide-included shifts in the NMR spectra of lanthanide (III) complexes derived from 1,4,7,10-tetrakis (N, N-diethylacetamido)-1,4,7,10-tetraazacyclododecane. *Inorganic Chemistry*, *34*, 3705–3715.

Gen, M., & Cheng, R. (1997). *Genetic algorithm and engineering design*. New York: Wiley.

Geoffrion, A. M., & Graves, G. W. (1976). Scheduling parallel production lines with changeover costs: Practical applications of a quadratic assignment/LP approach. *Operations Research*, *24*, 595–610.

Gilmore, P. C. (1962). Optimal and suboptimal algorithms for the quadratic assignment problem. *SIAM Journal on Applied Mathematics*, *10*, 305–313.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. New York: Addison-Wesley.

Goldberg, D. E., & Lingle, R. (1985). Alleles, loci and the travelling salesman problem. In: *J. J. Grefenstette (Ed.), Proceedings of the* 1$^{\text{st}}$ *international conference on genetic algorithms and their applications* (pp. 154–159). Lawrence Erlbaum Associates, Hilladale: NJ.

Hahn, P. M., Hightower, W. L., Johnson, T. A., Guignard-Spielberg, M., & Roucairol, C. (2001). Tree elaboration strategies in branch and bound algorithms for solving the quadratic assignment problem. *Yugoslavian Journal of Operational Research*, *11*(1), 41–60.

Heffley, D. R. (1980). Decomposition of the Koopmans-Beckmann problem. *Regional Science and Urban Economics*, *10*(4), 571–580.

Hubert, L. (1987). Assignment methods in combinatorial data analysis, statistics: Textbooks and monographs series, 73, Marcel Dekker.

Koopmans, T. C., & Beckmann, M. J. (1957). Assignment problems and the location of economic activities. *Econometrica*, *25*, 53–76.

Krarup, J., & Pruzan, P. M. (1978). Computer-aided layout design. *Mathematical Programming Study*, *9*, 75–94.

Lim, M. H., Yuan, Y., & Omatu, S. (2000). Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem. *Computational Optimization and Applications*, *15*, 249–268.

Louis, S. J., & Tang, R. (1999). Interactive genetic algorithms for the traveling salesman problem. In: *Proceedings of the genetic and evolutionary computing conference (GECCO)*, (pp. 385–392).

Merz, P., & Freisleben, B. (2000). Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, *4*, 337–352.

Michalewicz, Z. (1992). *Genetic algorithms + data structures = evolution programs*. Berlin: Springer-Verlag.

Migkikh, V. V., Topchy, A. A., Kureichik, V. M., & Tetelbaum, A. Y. (1996). Combined genetic and local search algorithm for the quadratic assignment problem. In: *Proceedings of the first international conference on evolutionary computation and its applications (EVCA'96)*, (pp. 335–341). Moscow: Presidium of the Russian Academy of Sciences.

Mills, P., Tsang, E., & Ford, J. (2003). Applying an extended guided local search to the quadratic assignment problem. *Annals of Operations Research*, *118*(1–4), 121–135.

Miranda, G., Luna, H. P. L., Mateus, G. R., & Ferreira, R. P. M. (2005). A performance guarantee heuristic for electronic components placement problems including thermal effects. *Computers and Operations Research*, *32*, 2937–2957.

Misevicius, A. (2004). An improved hybrid optimization algorithm for the quadratic assignment problem. *Mathematical Modelling and Analysis*, *9*(2), 149–168.

Misevicius, A., & Kilda, B. (2005). Comparison of crossover operators for the quadratic assignment problem. *Information Technology and Control*, *34*(2), 109–119.

Misevicius, A., & Rubliauskas, D. (2005). Performance of hybrid genetic algorithm for the grey pattern problem. *Information Technology and Control*, *34*(1), 15–24.

Oliver, I. M., Smith, D. J., & Holland, J. R. C. (1987). A study of permutation crossover operators on the travelling salesman problem. In: *J. J. Grefenstette (Ed.), Genetic algorithms and their applications, proceedings of the* 2nd *international conference on genetic algorithms* (pp. 224–230). Lawrence Erlbaum Associates, Hilladale: NJ.

Oliveira, C. A. S., Pardalos, M. P., & Resende, M. G. G. (2004). GRASP with path relinking for the quadratic assignment problem. In: *Experimental and efficient algorithms, third international workshop (WEA 2004), Brazil, LNCS, 3059*, (pp. 356–368). Springer.

Paul, G. (2011). An efficient implementation of the robust tabu search heuristic for sparse quadratic assignment problems. *European Journal of Operational Research*, *209*(3), 215–218.

Pollatschek, M. A., Gershoni, N., & Radday, Y. T. (1976). Optimization of the typewriter keyboard by simulation. *AngewandteI Informatik*, *17*, 438–439.

Puchinger, J., & Raidl, G. R. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A Survey and Classification. In: J. Mira and J. R. 'Alvarez (Eds.), *IWINAC 2005, LNCS 3562*, (pp. 41–53).

Sahni, S., & Gonzales, T. (1976). P-complete approximation problems. *Journal of the Association for Computing Machinery*, *23*, 555–565.

Serpell, M., & Smith, J. E. (2010). Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms. *Evolutionary Computation*, *18*(3), 491–514.

Steinberg, L. (1961). The backboard wiring problem: A placement algorithm. *SIAM Review*, *3*, 37–50.

Taillard, E. (1995). Comparison of iterative searches for the quadratic assignment problem. *Location Sciences*, *3*, 87–105.

Tate, D. E., & Smith, A. E. (1995). A genetic approach to the quadratic assignment problem. *Computers and Operations Research*, *22*, 73–83.

Vázquez, M., & Whitley, L. D. (2000). A hybrid genetic algorithm for the quadratic assignment problem. In L. D. Whitley, D. E. Goldberg, E. CantúPaz, et al. (Eds.), *Proceedings of the genetic and evolutionary computation conference (GECCO'00)* (pp. 135–142). San Francisco: Morgan Kaufmann.

Wilhelm, M. R., & Ward, T. L. (1987). Solving quadratic assignment problems by simulated annealing. *IEEE Transactions*, *19*, 107–119.

Zhang, H., Beltran-Royo, C., & Ma, L. (2013). Solving the quadratic assignment problem by means of general purpose mixed integer linear programming solvers. *Annals of Operations Research*, *207*, 261–278.