PERGAMON

Computers & Operations Research 27 (2000) 917–934

computers & operations research

www.elsevier.com/locate/dsw

# A greedy genetic algorithm for the quadratic assignment problem

Ravindra K. Ahuja[a], James B. Orlin[b,*], Ashish Tiwari[c]

[a]*Industrial and Systems Engineering Department, University of Florida, Gainesville, FL 32611, USA*
[b]*Operations Research Center, Massachusetts Institute of Technology, E40-149A, Cambridge, MA 02139, USA*
[c]*Department of Computer Science, State University of New York, Stony Brook, NY 11794, USA*

Received 1 March 1997; accepted 1 January 1999

**Abstract**

The Quadratic Assignment Problem (QAP) is one of the classical combinatorial optimization problems and is known for its diverse applications. In this paper, we suggest a genetic algorithm for the QAP and report its computational behavior. The genetic algorithm incorporates many greedy principles in its design and, hence, we refer to it as a *greedy genetic algorithm*. The ideas we incorporate in the greedy genetic algorithm include (i) generating the initial population using a randomized construction heuristic; (ii) new crossover schemes; (iii) a special purpose immigration scheme that promotes diversity; (iv) periodic local optimization of a subset of the population; (v) tournamenting among different populations; and (vi) an overall design that attempts to strike a balance between diversity and a bias towards fitter individuals. We test our algorithm on all the benchmark instances of QAPLIB, a well-known library of QAP instances. Out of the 132 total instances in QAPLIB of varied sizes, the greedy genetic algorithm obtained the best known solution for 103 instances, and for the remaining instances (except one) found solutions within 1% of the best known solutions.

**Scope and purpose**

Genetic Algorithms (GAs) are one of the most popular heuristic algorithms to solve optimization problems and is an extremely useful tool in an OR toolkit. For solving combinatorial optimization problems, GAs in their elementary forms are not competitive with other heuristic algorithms such as simulated annealing and tabu serach. In this paper, we have investigated the use of several possible enhancements to GAs and illustrated them using the Quadratic Assignment Problem (QAP), one of the hardest nut in the field of combinatorial optimization. Most of our enhancements use some greedy criteria to improve the quality of individuals in the population. We found that the overall performance of the GA for the QAP improves by

* Corresponding author. Tel.: + 1-352-392-3615; fax: + 1-352-392-3537.
*E-mail addresses:* ahuja@ufl.edu (R.K. Ahuja), jorlin@mit.edu (J.B. Orlin), astiwari@cs.sunysb.edu (A. Tiwari)

0305-0548/00/$ - see front matter 
PII: S0305-0548(99)00067-2

using greedy methods but not their overuse. Overuse of greedy methods adversely affects the diversity in the population. By striking a right balance between the greedy methods that improve the quality of the solution and the methods that promote diversity, we can obtain fairly effective heuristic algorithms for solving combinatorial optimization problems. 

*Keywords:* Genetic algorithms; Quadratic assignment problem; Heuristic algorithms; Local search algorithms

## 1. Introduction

The Quadratic Assignment Problem (QAP) is one of the classical combinatorial optimization problems and is widely regarded as one of the most difficult problem in this class. Given a set $N = \{1, 2, \ldots, n\}$, and $n \times n$ matrices $F = \{f_{ij}\}$, $D = \{d_{ij}\}$, and $C = \{c_{ij}\}$, the QAP is to find a permutation $\phi$ of the set $N$ which minimizes

$$z = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\phi(i)\phi(j)} + \sum_{i=1}^{n} c_{i\phi(i)}.$$

As an application of the QAP, consider the following campus planning problem. On a campus, new facilities are to be erected and the objective is to minimize the total walking distances for students and staff. Suppose there are $n$ available sites and $n$ facilities to locate. Let $d_{kl}$ denote the walking distance between the two sites $k$ and $l$ where the new facilities will be erected. Further, let $f_{ij}$ denote the number of people per week who travel between the facilities $i$ and $j$. Then, the decision problem is to assign facilities to sites so that the walking distance of people is minimized. Each assignment can be mathematically described by a permutation $\phi$ of $N = \{1, 2, \ldots, n\}$ such that $\phi(i) = k$ means that the facility $i$ is assigned to site $k$. The product $f_{ij} d_{\phi(i)\phi(j)}$ describes the weekly walking distance of people who travel between facilities $i$ and $j$. Consequently, the problem of minimizing the total walking distance reduces to identifying a permutation $\phi$ that minimizes the function $z$ defined above. This is an application of the QAP with each $c_{ik} = 0$. In this application, we have assumed that the cost of erecting a facility does not depend upon the site. In case it does, we will denote by $c_{ik}$ the cost of erecting facility $i$ at site $k$, and these costs will also play a role in determining the optimal assignment of facilities to sites.

Additional applications of QAP include (i) the allocation of plants to candidate locations; (ii) layout of plants; (iii) backboard wiring problem; (iv) design of control panels and typewriter keyboards; (v) balancing turbine runners; (vi) ordering of interrelated data on a magnetic tape; (vii) processor-to-processor assignment in a distributed processing environment; (viii) placement problem in VLSI design; (ix) analyzing chemical reactions for organic compounds; and (x) ranking of archaeological data. The details and references for these and additional applications can be found in Burkard [1], Malucelli [2], and Pardalos et al. [3].

On account of its diverse applications and the intrinsic difficulty of the problem, the QAP has been investigated extensively by the research community. The QAP has been proved to be an NP-complete problem, and a variety of exact and heuristic algorithms have been proposed. Exact

algorithms for QAP include approaches based on (i) dynamic programming [4]; (ii) cutting planes [5]; and (ii) branch and bound [6,7]. Among these, only branch and bound algorithms are guaranteed to obtain the optimum solution, but they are generally unable to solve problems of size larger than $n = 20$.

Since many applications of QAP give rise to problems of size far greater than 20, there is a special need for good heuristics for QAP that can solve large-size problems. Known heuristics for QAP can be classified into the following categories: (i) construction methods [8,9]; (ii) limited enumeration methods [10,11]; (iii) local improvement methods [12]; (iv) simulated annealing methods [13]; (v) tabu search methods [14,15]; and (vi) genetic algorithms [14–16]. Among these, (i) the tabu search method due to Skorin–Kapov [14] and (ii) the (randomized) local improvement method due to Li et al. [12] and Pardalos et al. [17], which they named as Greedy Randomized Adaptive Search Procedure (*GRASP*), are the two most accurate heuristic algorithms to solve the QAP.

In the past, genetic algorithms have been applied to a variety of combinatorial optimization problems (see, for example, Goldberg [18] and Davis [19]). In this paper, we suggest a genetic algorithm for the QAP which incorporates many ideas based on greedy principles. We call this algorithm the *greedy genetic algorithm*. The ideas we incorporate include (i) generating the initial population using a good (randomized) construction heuristic; (ii) new crossover schemes; (iii) special-purpose immigrations that promote diversity; (iv) periodic local optimization of a subset of the population; (v) tournamenting among different populations; and (vi) developing an overall design that attempts to strike a right balance between diversification and a bias towards fitter individuals. We test each of these ideas separately to assess its impact on the algorithm performance, and also compare our final algorithm on all benchmark instances in QAPLIB, a well-known library of QAP instances compiled by Burkard et al. [20]. We find the greedy genetic algorithm to be a very robust algorithm for QAP; out of the 132 total instances in QAPLIB of varied sizes, it obtained the best-known solution for 103 instances, and for the remaining instances (except one) found solutions within 1% of the best-known solutions. We also performed a limited computational testing comparing our genetic algorithm with GRASP and a genetic algorithm based on a representation developed by Bean [21].

In the past, three genetic algorithms for the QAP have been proposed. Bean [21] describes a generic genetic algorithm to solve those problems whose solutions are specified by permutations. This approach can be applied to the QAP as a special case but our computational investigations revealed this to be an unattractive approach since the deviation from the best-known solutions were too high. Tate and Smith [22] proposed another GA which uses the problem specific structure and tested it on 11 benchmark instances due to Nugent et al. [23] of size 5–30 facilities. This is a fairly direct implementation of the classical GA and does not use any greedy ideas. Fluerent and Ferland [16] describe another genetic algorithm which uses local search methods to improve the fitness of individuals. They present computational results on 13 benchmark instances of sizes from 40 to 100 facilities. They found that using local search methods substantially improves the correctness of the genetic algorithm for the QAP. In this paper, we investigate GAs incorporating some additional greedy ideas and test their effect on the accuracy of the solution. We also performed a more extensive computational testing which encompasses all the 132 benchmark instances in the QAPLIB, a well-known library of QAP instances, with problem sizes varying from 40 to 100.

## 2. Greedy genetic algorithm for QAP

Genetic algorithms represent a powerful and robust approach for developing heuristics for large-scale combinatorial optimization problems. The motivation underlying genetic algorithms can be expressed as follows: Evolution has been remarkably successful in developing complex and well-adapted species through relatively simple evolutionary mechanisms. A natural question is the following: What ideas can we adapt from our understanding of evolution theory so as to solve problems in other domains? This fundamental question has many different answers because of the richness of evolutionary phenomenon. Holland [24] provided the first answer to this question by developing genetic algorithms.

Genetic algorithms (GAs) imitate the process of evolution on an optimization problem. Each feasible solution of a problem is treated as an individual whose fitness is governed by the corresponding objective function value. A GA maintains a population of feasible solutions (also known as chromosomes) on which the concept of the survival of the fittest (among string structures) is applied. There is a structured yet randomized information exchange between two individuals (*crossover* operator) to give rise to better individuals. Diversity is added to the population by randomly changing some genes (*mutation* operator) or bringing in new individuals (*immigration* operator). A GA repeatedly applies these processes until the population converges.

GAs can be implemented in a variety of ways. The excellent books by Goldberg [18] and Davis [19] describe many possible variants of GAs. We also refer to these books for various GA notation such as chromosomes, alleles, genes, etc. In our research, we have used the GA with "steady-state reproduction" instead of the traditional "generational reproduction" because researchers have in general found it to be faster in practice (see, for example, Davis [19]). Our GA has the following high-level description:

```
algorithm genetic;
begin
    obtain initial population;
    repeat
        select two individuals I1 and I2 in the population;
        apply the crossover operator on I1 and I2 to produce a child I3;
        replace one of the two individuals I1 or I2 by I3;
        occasionally perform immigration;
    until the population converges;
end;
```

This is a simple high-level description of our GA. Each execution of the repeat loop is called a *trial*. We will now describe how various steps of the GA are implemented when specialized for the QAP. We have incorporated several heuristics based on greedy principles to improve the performance of the GA. We will also describe the details of these heuristics.

### 2.1. Encoding scheme

An encoding scheme maps feasible solutions of the problem to strings. The effectiveness of the crossover operator depends greatly on the encoding scheme used. The encoding should be such

that the crossover operator preserves high performing partial arrangements (schemata) of strings, and minor changes in the chromosome translate into minor changes in the corresponding solution. For the QAP, a natural encoding is the permutation of $n$ numbers in the set $N = \{1, 2, \ldots, n\}$, where the $j$th number in the permutation denotes the facility located on site $j$. In our implementation, we have used this encoding scheme.

### 2.2. Initial population generation

The performance of a GA is often sensitive to the quality of its initial population. The "goodness" of the initial population depends both on the average fitness (that is, the objective function value) of individuals in the population and the diversity in the population. Losing on either count tends to produce a poor GA. By having an initial population with better fitness values, we typically get better final individuals. Further, high diversity in the population inhibits early convergence to a locally optimal solution. We used a population size of 100 individuals.

For the QAP, several heuristics can be used to generate the initial population. In our implementation, we have used the initial solutions produced by GRASP (see [12]). GRASP consists of two phases: a construction phase and an improvement phase. The construction phase uses a randomized greedy algorithm to assign facilities to locations one-by-one, at each step minimizing the total cost with respect to the assignments already made. The improvement phase uses a neighborhood search technique to improve the solution obtained by the first phase iteratively. Each application of GRASP yields a (possibly) different solution because of the randomization used in the construction phase. In our implementation of the GA, we have used the initial population comprising of solutions produced by the construction phase of GRASP. We did not apply the improvement phase. Our preliminary computational tests suggested that improving the initial population through the improvement phase would not have led to better overall results.

### 2.3. Selection

The selection criteria is used to select the two parents to apply the crossover operator. The appropriateness of a selection criteria for a GA depends upon the other GA operators chosen. In the literature, a typical selection criteria gives a higher priority to fitter individuals since this leads to a faster convergence of the GA. We compared this selection criteria with the criteria where each individual is equally likely to be selected. We obtained better overall results when the selection is not biased towards fitter individuals. Consequently, in our GA we selected both parents randomly giving equal probability of selection to each individual in the population.

### 2.4. Crossover

The crossover scheme is widely acknowledged as critical to the success of GA. The burden of "exploration" and "exploitation" falls upon it simultaneously. The crossover scheme should be capable of producing a new feasible solution (i.e., a child) by combining good characteristics of both parents. Preferably, the child should be considerably different from each parent. We observed that when the initial population is generated using a good heuristic method, then standard crossover schemes, such as the order crossover given in Davis [19], generally fail to produce individuals

better than both the parents. As the order crossover scheme did not give good results, we tested two more sophisticated crossover schemes, called the (i) path crossover scheme, and (ii) optimized crossover scheme. We shall now describe these schemes in more detail. We remark that these crossover schemes are not specific to the QAP, and would apply to any problem with chromosomes represented by permutations.

### 2.5. Path crossover

Let $I_1$ and $I_2$ denote two individuals. Applying a crossover scheme to these two parents produces a child, which we denote by $I_3$. Let the chromosomes of $I_1$, $I_2$, and $I_3$ be represented by $a_1 - a_2 - \cdots - a_n$, $b_1 - b_2 - \cdots - b_n$, and $c_1 - c_2 - \cdots - c_n$, respectively. In any crossover scheme that we consider, the child inherits any genes common to both the parents; that is, if $a_k = b_k$ for some $k$, then $c_k = a_k = b_k$. In the two types of path crossovers described next, we define a path $p_1 - p_2 - \cdots - p_r$ where (i) each node in the path corresponds to an individual; (ii) the first node $p_1$ corresponds to $I_1$, the last node $p_k$ corresponds to $I_2$, and, (iii) for each $1 \leqslant k \leqslant r - 1$, $p_{k+1}$ is obtained from $p_k$ by performing a primitive transformation. We considered two types of primitive transformations: *swap* and *insert*. If we use a swap transformation, we get the *swap path crossover* scheme. If we use an insert transformation, we get the *insert path crossover* scheme. The path crossover was originally developed by Glover [25], who referred to it as "path relinking". We use the alternative term "path crossover", because we think that it is more consistent with the usual terminology used in genetic algorithms.

In the swap path crossover scheme, we start at some position of the chromosomes, which is determined using a random process, and examine the chromosomes corresponding to $I_1$ and $I_2$ from left to right in a cyclic fashion. If the alleles at the position (or gene) being looked at are the same, we move to the next position; otherwise, we perform a swap of the alleles of two genes in $I_1$ or in $I_2$, whichever gives the fitter solution, so that the alleles at the position being looked at become alike. We repeat this process until all alleles have been considered. All chromosomes obtained using this process are valid children of $I_1$ and $I_2$; out of them we let $I_3$ denote the fittest child.

We illustrate the swap path crossover in Fig. 1(a). Suppose we start at the first position. In parent 1, facility 5 is located at the first site and in parent 2 facility 2 is located at this site. There are two ways in which the two parents can move closer to one-another; by swapping the sites of the facilities 2 and 5 in parent 1 or in parent 2. We compute the objective function values of these two solutions and we perform the swap for which the corresponding solution has a lower cost; the resulting solution is the first child. We then consider the alleles in the two resulting solutions starting at the second position and obtain possibly several children; the best among these is $I_3$. In this example, we start at the first position; in general, we start at a position selected randomly and examine subsequent positions from left to right in a wrap-around fashion until all positions have been considered.

The insert transformation is similar to the swap transformation; the only difference is that to have the same alleles at a given position in the two chromosomes, we insert an allele at a new place thereby shifting the other alleles to the right in a way so that common alleles do not move. We illustrate the insert path crossover in Fig. 1(b). For parent 1, the insert transformation consists of removing facility 2 from site 2 and moving to site 1 and facility 5 shifts to site 2. For parent 2, this transformation causes bigger changes. It moves facility 5 to site 1, facility 2 moves to site 2,

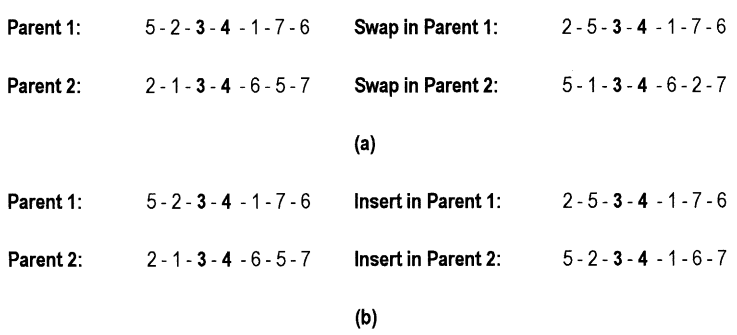


Fig. 1. Illustrating swap and insert path crossovers. The common alleles are shown in bold type.

facility 1 moves to site 5 (because facilities at sites 3 and 4 are fixed), and facility 6 moves to site 6. It is easy to see that the insert transformation causes changes which are radically different than the swaps.

Use of the swap or insert path crossover scheme yields one child $I_3$. One possible scheme is to keep the best two individuals among the two parents ($I_1$ and $I_2$) and the one child ($I_3$). We found that this greedy scheme leads to premature convergence. If the parents are very good, the child is usually less fit than both parents, and the crossover operation does not change the population. Rather, we required the child to replace one of the parents. If the child is fitter than both the parents, then it replaces the parent with which it is more similar; otherwise, the child replaces the worse parent. This crossover scheme ensures that the children not only inherit useful information from their parents but also search new regions of the solution space. Further, our rule for replacement of a parent by the child ensures that the best individual in the population is never replaced. Such an approach is called as *elitist* approach in the literature.

### 2.6. Optimized crossover

In the optimized crossover scheme, the genes of the parents are selected so as to optimize the fitness of the child. Use of optimized crossover in GAs has yielded encouraging results for the independent set problem [26].

Using the optimized crossover scheme, the two parents produce a child, which is called the *optimized child*. We will now explain how to determine an optimized child for QAP. Let $I_1 = a_1 - a_2 - \cdots - a_n$ and $I_2 = b_1 - b_2 - \cdots - b_n$ be two parents. An optimized child of $I_1$ and $I_2$ is an individual $I_3 = c_1 - c_2 - \cdots - c_n$ (i) which satisfies $c_i = a_i$ or $b_i$ for all $1 \leqslant i \leqslant n$, and (ii) has the smallest objective function value among individuals satisfying (i). To understand how to obtain the optimized child, a network formulation will be helpful. Each solution of a QAP can be represented by a perfect matching in a complete undirected bipartite graph $G = (U \cup V, U \times V)$ with $U = \{u_1, u_2, \ldots, u_n\}$ representing facilities and $V = \{v_1, v_2, \ldots, v_n\}$ representing sites. We can represent an individual I by an arc set $E$ in $G$ as follows : $(u_i, v_j) \in E$ if and only if facility $i$ is located at site $j$. Now, consider the union of two arc sets $E_1$ and $E_2$ corresponding to two parents $I_1$ and $I_2$. An optimized child of $I_1$ and $I_2$ is a perfect matching in $G' = (U \cup V, E_1 \cup E_2)$ with the least

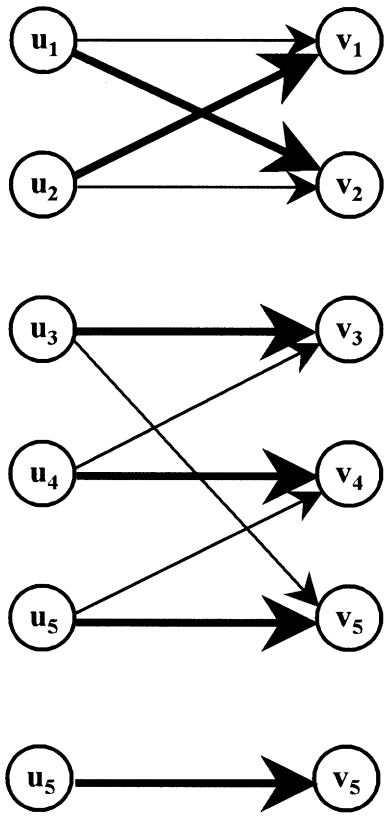


Fig. 2. Illustrating the optimized crossover. There are two cycles and four candidates for the optimized child.

objective function value. See, for example, Fig. 2. $E_1 \cup E_2$ is a collection of (i) singleton arcs (corresponding to common alleles), and (ii) node-disjoint cycles (corresponding to different alleles). Suppose that there are $k$ cycles. In this case, there are exactly $2^k$ perfect matchings in $G'$, because once we select any arc in any node-disjoint cycle for inclusion in the matching, we must select every other alternate arc. Hence arcs in any cycle can be selected in exactly two ways. Since there are $k$ cycles, there will be exactly $2^k$ children of $I_1$ and $I_2$; for each such child, we determine the objective function value, and the child with the least objective function value is the optimized child. This child replaces one of the parents using the same strategy as in the path crossover scheme. One might wonder whether we can identify an optimized child in polynomial time. This is not likely because it can be shown (proof omitted) that determining an optimized child is also an NP-hard problem. Nevertheless, the exhaustive enumeration method may be efficient in practice because $k$ may be very small. For most problems in our test bed, the number of cycles is less than 5.

In our investigations, we used all three crossover schemes outlined above. In preliminary testing, we obtained best results from the insert path crossover scheme. Subsequently, we performed detailed computational testing using the insert path crossover scheme.

### 2.7. Mutation/Immigration

Mutation refers to the process of increasing diversity in the population by introducing random variations in the members of the population. Researchers have used immigration in place of mutation to achieve the same purpose (see, for example, Davis [19]). Immigration refers to the process of replacing poor members of the current population by bringing in new individuals. For our implementation of GA, we choose immigration. We also used a variant of mutation, which we discuss below in the subsection on local optimization.

We could use the same method to generate immigrants that we used to generate the initial population. We found that this did not introduce new genetic material to the population and the GA converged prematurely, possibly because the method did not introduce sufficient new genetic material. Therefore, we use a different scheme to generate immigrants that produces individuals significantly different than the ones already in the population and hence forces the algorithm to search newer regions of solution space.

Let us define an $n \times n$ matrix $\alpha = \{\alpha_{ij}\}$ whose $(i, j)$th position stores the number of individuals in the population (present as well as all past members) in which facility $j$ has been assigned to site $i$. A lower value of $\alpha_{ij}$ implies that fewer individuals in the population had facility $i$ assigned to site $j$. To promote diversity, we encourage this assignment in new individuals. We generate a new individual for immigration using the following method. We first select a permutation of $n$ numbers obtained using a random process; this gives us the order in which we consider the sites one-by-one. To each site under consideration, say site $k$, we assign the unassigned facility $l$ for which $\alpha_{kl}$ is minimum. We stop when all sites have been considered; at this point, all facilities have also been assigned. This assignment promotes individuals from the under-explored solution space, and though the process does not explicitly use randomness, it typically yields new individuals.

We experimented with three rates of immigration: 10% (that is, one immigration after every 10 trials), 20%, and variable immigration rate. In the variable immigration rate, we start with the immigration rate of 10% and increase this rate by 2% after every 200 trials. We also introduced immigration when we found that the diversity in the population is below an accepted threshold level.

### 2.8. Local optimization

Commonly used approaches for solving NP-hard problems are local optimization, simulated annealing, and tabu search; all three approaches rely on a neighborhood structure. For the QAP problem, the 2-exchange neighborhood (that is, obtained by swapping two numbers in the permutation) has been extensively investigated and found to be a fairly powerful heuristic. Local optimization based upon the 2-exchange neighborhood can be used to improve the quality of any population. We observed that periodic local optimization of a subset of the population improves the performance of the GA, and hence we incorporated it into the algorithm. After every 200 trials, we locally optimize 20% of the population. We store individuals using arrays: after 200 trials we locally optimize the first 20%, after 400 trials we locally optimize the next 20%, and so on. When we reach the end of the array, we return to the beginning of the array.

### 2.9. Tournamenting

In principle, one can apply a GA many times starting with different populations, and choose the best individual obtained among all the runs. As an alternative, we consider the following approach, one that takes the final population of two different runs, keeps 50% of the individuals in the two runs, and applies the GA again with this mixed population as the initial population. The basic idea behind this scheme is that it allows the best alleles of two different runs to be combined through crossover operator to create fitter individuals. In a sense it permits the GA to search regions that are "in between" the regions to which the two populations converged. Generalizing this idea further for multiple runs gives us a GA with "tournamenting". Fig. 3 describes the tournamenting scheme with four teams; one can easily generalize this approach to a larger number of teams.

When we use tournamenting, we terminate the execution of a GA before the population converges, since we found that allowing the GA to run until the population converges leads to poorer solutions, possibly because of the lack of diversity in the final populations. We found that after 1000 trials of our GA, the population quality is good and there is sufficient diversity. We choose to terminate the GA after 1000 trials.

We considered several options for eliminating 50% of the two populations, say $P'_1$ and $P'_2$ in order to obtain the initial population $P_{12}$ for the next round, including the following two options: (i) select the best 50% of the union of the two populations; and (ii) the union of the best 50% of the populations $P'_1$ and $P'_2$. Both these schemes have the drawback that multiple copies of the fitter

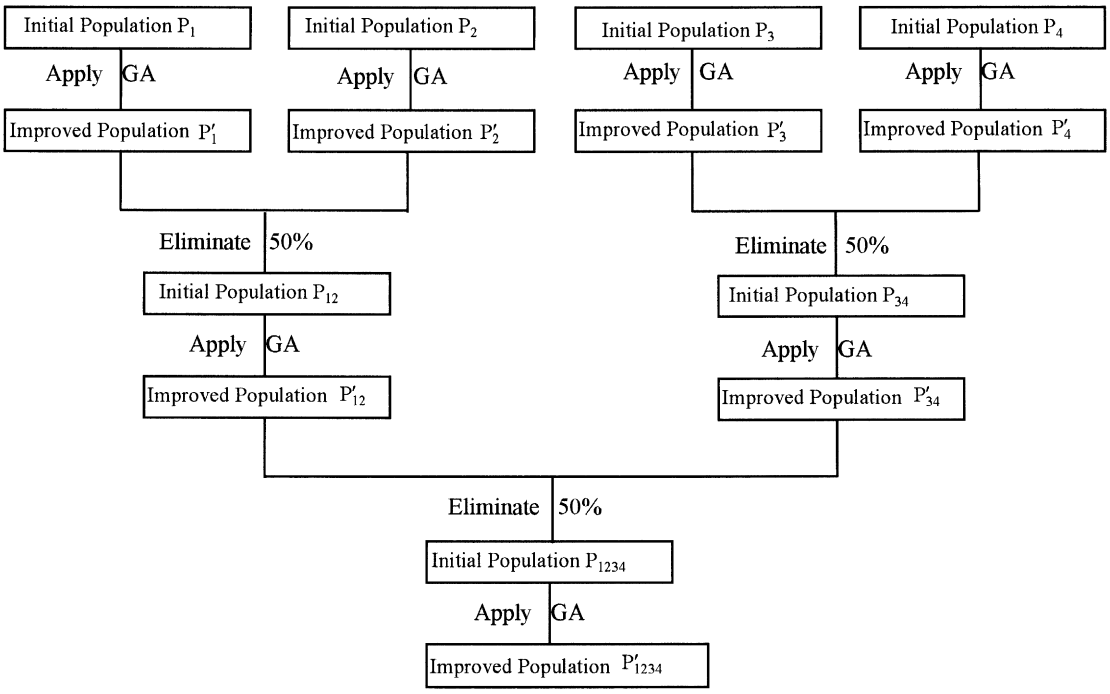


Fig. 3. GA with tournamenting with four teams.

individuals might dominate the population. We obtained a better mix by performing a one-to-one competition between individuals of the two populations; we compare the $k$th individual of $P'_1$ with the $k$th individual of $P'_2$ and the fitter one moves to the next round.

## 3. Computational results

In this section, we present the computational results of the greedy GA when applied to all the 132 instances of the QAPLIB compiled by Burkard et al. [20]. Our algorithms were programmed in the C programming language and implemented on a HP 6000 computer. We also compared our algorithm with GRASP, and our implementation of a random key genetic algorithm due to Bean [4].

### 3.1. Computational time and accuracy

We present in Table 1 the running times and the accuracy of the greedy GA applied to all the instances in QAPLIB. The computational times given in Table 1 are in seconds. Problem sizes in QAPLIB vary from 10 to 100, and all problems are dense. The middle numeric digits in the problem identification state the problem size. We present results for the following three versions of the algorithm:

*Version* 1: The initial population is generated using Phase 1 of GRASP, all individuals in the population are equally likely to be selected for crossover; insert path crossover is used; immigration rate is set to 10%; after every 200 trials, 20% of the population is locally optimized; and tournamenting with four teams is used.

*Version* 2: Same as Version 1 except that the immigration rate is variable. The algorithm starts with the immigration rate of 10%, and after every 200 trials the immigration rate is increased by 2%.

*Version* 3: Same as Version 2 except that the tournamenting with eight teams is used.

The objective function values of the best known solutions are given in Burkard et al. [20]. We will give only the percent deviation from the best known solutions, in the columns marked GA-1, GA-2, and GA-3. We give the computational times only for version 1; the time taken by version 2 is comparable to the time taken by version 1, and version 3 takes time approximately twice the times taken by version 1. We observe that the average deviations of these three versions are 45%, 26%, and 11%, respectively. Thus, the greedy GA with eight-team tournamenting was the most accuate method for this test bed. It also has a very robust performance. We find that out of 132 problem instances, this method obtained the best-known solution for 103 problem instances; and out of the remaining 29 problems only once was the deviation more than one percent from the best-known solution. Moreover, we obtained these results by applying the algorithm only once. We anticipate that the results will improve if we apply the greedy GA several times starting with different seeds for the random number generator.

We now present results contrasting the various schemes employed by us in the greedy GA. We select Version 1 of the greedy GA (without tournamenting) as the benchmark algorithm and investigate how its performance changes as we modify its attributes (such as, the selection scheme,

Table 1
Computational results of the three versions of GAs

| Problem name | GA-1 | TIME | GA-2 | GA-3 | Problem name | GA-1 | TIME | GA-2 | GA-3 |
|---|---|---|---|---|---|---|---|---|---|
| bur26a.dat | 0.00 | 117.3 | 0.00 | 0.00 | esc16h.dat | 0.00 | 24.0 | 0.00 | 0.00 |
| bur26b.dat | 0.00 | 112.7 | 0.00 | 0.00 | esc16i.dat | 0.00 | 25.8 | 0.00 | 0.00 |
| bur26c.dat | 0.00 | 113.5 | 0.00 | 0.00 | esc16j.dat | 0.00 | 201.1 | 0.00 | 0.00 |
| bur26d.dat | 0.00 | 106.7 | 0.00 | 0.00 | esc32a.dat | 3.08 | 190.9 | 0.00 | 0.00 |
| bur26e.dat | 0.00 | 109.1 | 0.00 | 0.00 | esc32b.dat | 0.00 | 200.1 | 0.00 | 0.00 |
| bur26f.dat | 0.00 | 102.2 | 0.00 | 0.00 | esc32c.dat | 0.00 | 194.6 | 0.00 | 0.00 |
| bur26g.dat | 0.00 | 97.1 | 0.00 | 0.00 | esc32d.dat | 0.00 | 176.3 | 0.00 | 0.00 |
| bur26h.dat | 0.00 | 101.9 | 0.00 | 0.00 | esc32e.dat | 0.00 | 184.9 | 0.00 | 0.00 |
| car10ga.dat | 0.00 | 3.9 | 0.00 | 0.00 | esc32f.dat | 0.00 | 184.4 | 0.00 | 0.00 |
| car10gb.dat | 0.00 | 3.9 | 0.00 | 0.00 | esc32g.dat | 0.00 | 185.5 | 0.00 | 0.00 |
| car10gc.dat | 0.00 | 4.3 | 0.00 | 0.00 | esc32h.dat | 0.00 | 174.5 | 0.00 | 0.00 |
| car10gd.dat | 0.00 | 4.2 | 0.00 | 0.00 | esc64a.dat | 0.00 | 1315.4 | 0.00 | 0.00 |
| car10ge.dat | 0.00 | 4.0 | 0.00 | 0.00 | kra30a.dat | 1.34 | 150.7 | 1.57 | 0.00 |
| car10gf.dat | 0.00 | 4.4 | 0.00 | 0.00 | kra30b.dat | 0.18 | 165.3 | 0.08 | 0.00 |
| car10gg.dat | 0.00 | 4.3 | 0.00 | 0.00 | lipa10a.dat | 0.00 | 4.8 | 0.00 | 0.00 |
| car10gh.dat | 0.00 | 4.0 | 0.00 | 0.00 | lipa10b.dat | 0.00 | 4.5 | 0.00 | 0.00 |
| car10gi.dat | 0.00 | 3.9 | 0.00 | 0.00 | lipa20a.dat | 0.00 | 37.4 | 0.00 | 0.00 |
| car10gj.dat | 0.00 | 4.1 | 0.00 | 0.00 | lipa20b.dat | 0.00 | 37.2 | 0.00 | 0.00 |
| car10gk.dat | 0.00 | 3.8 | 0.00 | 0.00 | lipa30a.dat | 0.00 | 172.3 | 0.00 | 0.00 |
| car10gl.dat | 0.00 | 4.9 | 0.00 | 0.00 | lipa30b.dat | 0.00 | 168.6 | 0.00 | 0.00 |
| car10gm.dat | 0.00 | 4.2 | 0.00 | 0.00 | lipa40a.dat | 0.00 | 510.9 | 1.10 | 0.96 |
| car10gn.dat | 0.00 | 4.3 | 0.00 | 0.00 | lipa40b.dat | 0.00 | 513.2 | 0.00 | 0.00 |
| car10go.dat | 0.00 | 3.9 | 0.00 | 0.00 | lipa50a.dat | 0.95 | 743.1 | 0.93 | 0.95 |
| car10pa.dat | 0.00 | 4.6 | 0.00 | 0.00 | lipa50b.dat | 0.00 | 754.7 | 0.00 | 0.00 |
| car10pb.dat | 0.00 | 4.3 | 0.00 | 0.00 | lipa60a.dat | 0.90 | 1528.5 | 0.82 | 0.77 |
| car10pc.dat | 0.00 | 4.3 | 0.00 | 0.00 | lipa60b.dat | 0.00 | 1523.7 | 0.00 | 0.00 |
| car10pd.dat | 0.00 | 4.4 | 0.00 | 0.00 | lipa70a.dat | 0.72 | 3074.2 | 0.73 | 0.71 |
| car10pe.dat | 0.00 | 3.9 | 0.00 | 0.00 | lipa70b.dat | 0.00 | 3061.6 | 0.00 | 0.00 |
| car10pf.dat | 0.00 | 4.5 | 0.00 | 0.00 | lipa80a.dat | 0.65 | 4759.5 | 0.67 | 0.61 |
| car10pg.dat | 0.00 | 4.5 | 0.00 | 0.00 | lipa80b.dat | 19.69 | 4749.4 | 0.00 | 0.00 |
| car10ph.dat | 0.00 | 4.2 | 0.00 | 0.00 | lipa90a.dat | 0.59 | 6179.2 | 0.62 | 0.58 |
| car10pi.dat | 0.00 | 4.0 | 0.00 | 0.00 | lipa90b.dat | 0.00 | 6159.7 | 0.00 | 0.00 |
| car10pj.dat | 0.00 | 4.6 | 0.00 | 0.00 | nug05.dat | 0.00 | 1.2 | 0.00 | 0.00 |
| car10pk.dat | 0.00 | 4.4 | 0.00 | 0.00 | nug06.dat | 0.00 | 1.8 | 0.00 | 0.00 |
| car10pl.dat | 0.00 | 4.6 | 0.00 | 0.00 | nug07.dat | 0.00 | 2.4 | 0.00 | 0.00 |
| car10pm.dat | 0.00 | 4.4 | 0.00 | 0.00 | nug08.dat | 0.00 | 3.4 | 0.00 | 0.00 |
| car10pn.dat | 0.00 | 4.1 | 0.00 | 0.00 | nug12.dat | 0.00 | 9.5 | 0.00 | 0.00 |
| car10po.dat | 0.00 | 3.8 | 0.00 | 0.00 | nug15.dat | 0.00 | 20.7 | 0.00 | 0.00 |
| chr12a.dat | 0.00 | 9.8 | 0.00 | 0.00 | nug20.dat | 0.00 | 48.9 | 0.00 | 0.00 |
| chr12b.dat | 0.00 | 9.2 | 0.00 | 0.00 | nug30.dat | 0.07 | 177.1 | 0.07 | 0.07 |
| chr12c.dat | 0.00 | 10.1 | 0.00 | 0.00 | rou10.dat | 0.00 | 5.1 | 0.00 | 0.00 |
| chr15a.dat | 0.00 | 20.3 | 0.00 | 0.40 | rou12.dat | 0.00 | 9.8 | 0.00 | 0.00 |
| chr15b.dat | 0.00 | 20.9 | 0.00 | 0.00 | rou15.dat | 0.00 | 17.3 | 0.00 | 0.00 |
| chr15c.dat | 0.00 | 22.0 | 4.59 | 0.00 | rou20.dat | 0.08 | 37.6 | 0.20 | 0.16 |
| chr18a.dat | 0.00 | 39.5 | 0.18 | 0.40 | scr10.dat | 0.00 | 5.3 | 0.00 | 0.00 |
| chr18b.dat | 0.00 | 39.4 | 0.00 | 0.00 | scr12.dat | 0.00 | 9.4 | 0.00 | 0.00 |
| chr20a.dat | 0.18 | 47.3 | 7.21 | 0.00 | scr15.dat | 0.00 | 17.6 | 0.00 | 0.00 |

Table 1 (*continued*)

| Problem name | GA-1 | TIME | GA-2 | GA-3 | Problem name | GA-1 | TIME | GA-2 | GA-3 |
|---|---|---|---|---|---|---|---|---|---|
| chr20b.dat | 7.40 | 48.2 | 3.48 | 5.13 | scr20.dat | 0.03 | 39.8 | 0.03 | 0.00 |
| chr20c.dat | 4.72 | 48.9 | 0.00 | 0.00 | sko42.dat | 0.23 | 503.1 | 0.00 | 0.25 |
| chr22a.dat | 0.62 | 72.9 | 0.62 | 0.75 | sko49.dat | 0.27 | 626.1 | 0.27 | 0.21 |
| chr22b.dat | 1.19 | 76.1 | 2.87 | 0.00 | sko56.dat | 0.08 | 1488.0 | 0.07 | 0.02 |
| chr25a.dat | 10.54 | 96.8 | 12.86 | 0.00 | sko64.dat | 0.38 | 1894.1 | 0.17 | 0.22 |
| els19.dat | 0.00 | 40.3 | 0.00 | 0.00 | sko72.date | 0.44 | 2539.0 | 0.27 | 0.29 |
| esc08a.dat | 0.00 | 3.5 | 0.00 | 0.00 | sko81.dat | 0.23 | 5482.1 | 0.40 | 0.20 |
| esc08b.dat | 0.00 | 3.7 | 0.00 | 0.00 | sko90.dat | 0.43 | 6348.9 | 0.33 | 0.27 |
| esc08c.dat | 0.00 | 3.8 | 0.00 | 0.00 | sko100a.dat | 0.19 | 8304.1 | 0.30 | 0.21 |
| esc08d.dat | 0.00 | 3.6 | 0.00 | 0.00 | sko100b.dat | 0.48 | 7364.7 | 0.24 | 0.14 |
| esc08e.dat | 0.00 | 3.6 | 0.00 | 0.00 | sko100c.dat | 0.01 | 10157.1 | 0.27 | 0.20 |
| esc08f.dat | 0.00 | 3.7 | 0.00 | 0.00 | sko100d.dat | 0.35 | 10151.1 | 0.38 | 0.17 |
| esc16a.dat | 0.00 | 23.7 | 0.00 | 0.00 | sko100e.dat | 0.23 | 10563.5 | 0.24 | 0.24 |
| esc16b.dat | 0.00 | 24.1 | 0.00 | 0.00 | sko100f.dat | 0.19 | 10739.5 | 0.32 | 0.29 |
| esc16c.dat | 0.00 | 26.7 | 0.00 | 0.00 | ste36a.dat | 1.47 | 354.8 | 1.39 | 0.27 |
| esc16d.dat | 0.00 | 26.6 | 0.00 | 0.00 | tho30.dat | 0.31 | 197.8 | 0.39 | 0.00 |
| esc16e.dat | 0.00 | 23.4 | 0.00 | 0.00 | tho40.dat | 0.33 | 479.1 | 0.20 | 0.32 |
| esc16f.dat | 0.00 | 23.0 | 0.00 | 0.00 | wil50.dat | 0.09 | 1057.6 | 0.10 | 0.07 |
| esc16g.dat | 0.00 | 24.9 | 0.00 | 0.00 | wil100.dat | 0.18 | 10271.9 | 0.21 | 0.20 |

or the crossover). For the sake of brevity, we do not present results for modifications made in two or more schemes simultaneously; we tried some of these modifications, and found them to be by and large consistent with modifications in single scheme. We investigated the algorithmic modifications using the following 11 benchmark instances selected from QAPLIB:

| Prob. No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Prob. Id. | kra30b | lipa50a | lipa60a | lipa70a | lipa80a | sko49 | sko90 | sko100a | tho30 | wil50 | wil100 |

*Initial Population*: We compared GAs using the following three methods of generating the initial population: (i) random population; (ii) using the first phase of GRASP; and (iii) using both phases of GRASP. These methods give average deviations of 0.73%, 0.7%, and 0.75%, respectively. We observed from this limited computational testing that the performance of the GA is relatively insensitive to the method used for initial population generation.

*Selection*: We next compared two selection strategies: when all individuals are equally likely to be selected for crossover, and when fitter individuals have greater likelihood for selection (the most fit individual is twice as likely to be selected as the least fit individual). We found that both the methods have an average deviation of 0.69%. Hence, the unbiased selection appears to be as good as the biased selection.

*Crossover*: We next determined the % deviation for the best individual obtained by GA using three different crossover schemes: optimized crossover, swap path crossover, and insert path

crossover. We found that these crossover schemes obtained average deviations of 0.92%, 0.76%, and 0.69%, respectively. For this limited testing, the insert path crossover gives the best results among the three methods.

*Immigration*: The performance of a GA can depend upon the method used to generate immigrants. We compared two schemes of generating the immigrants – the first scheme generates immigrants using the same method as used to generate the initial population and second scheme using a different method that promotes diversity (see Section 2). The GA with the first scheme gave an average deviation of 0.82% and the GA with the second scheme gave an average deviation of 0.63%. Hence the scheme that promotes diversity appears to give slightly better overall results for this test bed.

*Local optimization*: In Fig. 4, we assess the effect of periodically optimizing a part of the population. The GA without local optimization has an average deviation of 6.4% and the GA with local optimization has an average deviation of 0.6%. Clearly, the GA in which local optimization is performed is much more accurate than the GA in which local optimization is not used. We see that the local optimization plays an important role in improving the quality of the individuals in the population. In fact, the local optimization contributes more to the quality of the solution than any of the other features that we analysed.

*Tournamenting*: In Fig. 5, we compare the percent deviations of the best objective function value obtained using the GA without tournamenting, and two GAs with tournamenting (4 and 8 teams). We found that these methods obtain average deviations of 0.78%, 0.37%, and 0.34%, respectively. Hence tournamenting was successful in improving the quality of the solutions for our test bed. But this advantage was at the expense of an increased running time. A GA with four-team tornamenting applies GA 7 times and the GA with eight-team tournamenting applies GA 15 times. To perform a fairer comparison, we compared the GA with four-team tournamenting with the best of seven runs of GA without tournamenting and found that GA with tournamenting is not significantly better than the GA without tournamenting. Hence, after accounting for time differences tournamenting was not beneficial.

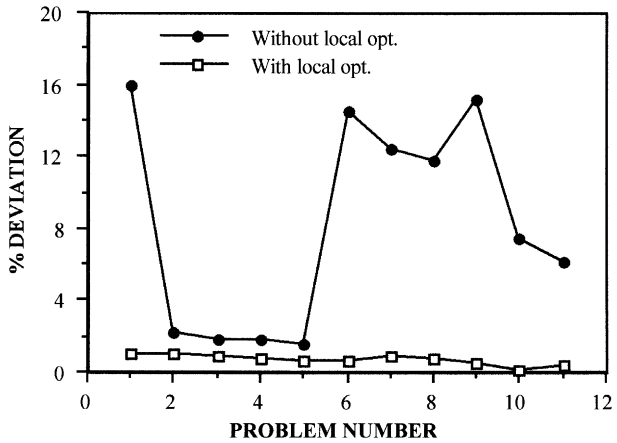


Fig. 4. Effect of local optimization on the performance of GA.

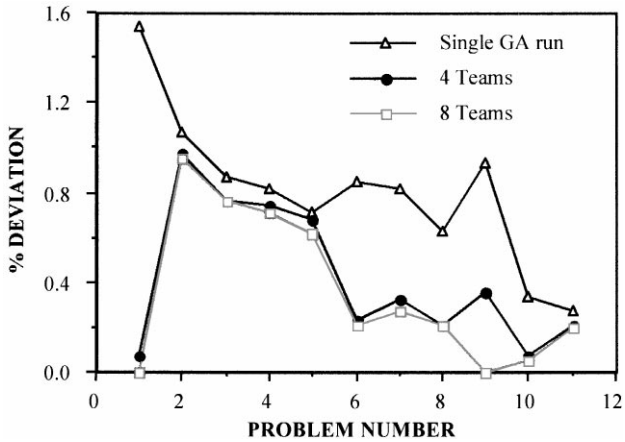


Fig. 5. Effect of tournamenting on the performance of GA.

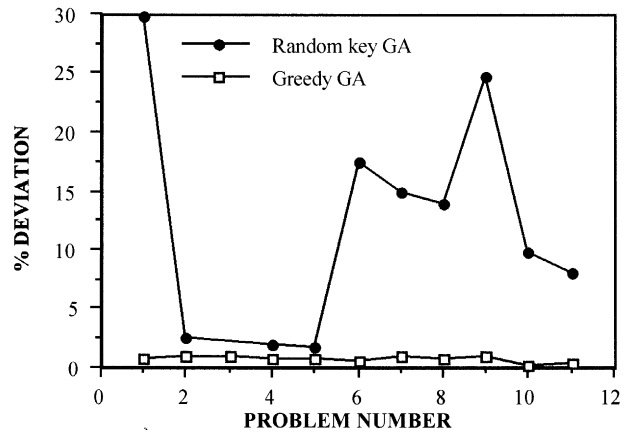


Fig. 6. Comparing greedy GA with random keys GA.

*Comparison with Random keys Algorithm*: Bean [21] gives a general purpose GA for optimization problems with a permutation encoding, called the *random keys GA*. This approach applies to QAP as well. We developed a computer program for this algorithm and compared it to the greedy GA on the benchmark instances. Fig. 6 presents these results. The average deviation of the greedy GA was less than 1% and the average deviation of the random keys GA was about 11%. Hence we found the greedy GA to be substantially superior to the random keys GA in terms of the quality of the solution obtained. We note that the random key GA is a generic approach and was not fine-tuned for the QAP. In contrast, the greedy GA incorporates a number of features that exploit the structure of the QAP.

*Comparison with GRASP*: Li et al. [12] developed GRASP and presented extensive computational results that suggested that GRASP is the most accurate method for solving QAP for medium- to large-size problems. GRASP is an iterative method and is applied a large number of

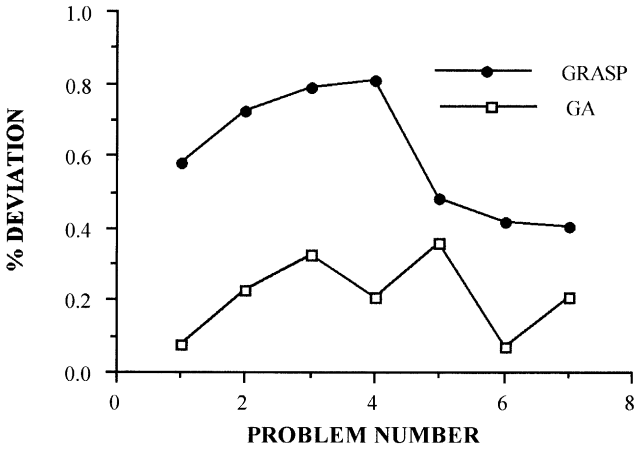


Fig. 7. Comparison of greedy GA with GRASP.

times with (possibly) different starting solutions. The accuracy of the best solution improves as more iterations are performed. In a similar fashion, the greedy GA can be applied several times to improve the performance of the algorithm. To obtain a fair comparison between GRASP and GA, we compared the accuracy of the two algorithms while running GRASP for the same time as that taken by the greedy GA. We apply GA with tournamenting (four teams) once and note the time taken, say $x$ s. We then run GRASP for $x$ s and compare the best solution obtained with the GA solution. Fig. 7 gives this comparison on the benchmark instances. We could not do the comparison on all the 11 benchmark problem instances because four of these instances are nonsymmetric and GRASP can be applied to symmetric instances only. On these seven benchmark instances, the greedy genetic algorithm performs better than GRASP. This computational testing is suggestive and by no means conclusive. First of all, we tested on a limited number of test instances. Secondly, we devoted considerable effort in fine-tuning the genetic algorithm but made little effort to fine-tune the parameter settings for GRASP.

## 4. Summary and conclusions

In this paper, we developed a new genetic algorithm for QAP and presented computational results of the algorithm on a large set of standard problems in the literature. We call our algorithm the *greedy genetic algorithm* since it incorporates many greedy principles. The principal findings of our paper are the following:

1. Path crossover performed slightly better than optimized crossover, possibly because it keeps the population sufficiently diverse.
2. The algorithm had slightly better performance when the method used to generate immigrants was different than the method used to generate the initial population.
3. Periodically optimizing a subset of the population improves the overall performance of the genetic algorithm substantially.

We find that success of a greedy genetic algorithm depends upon striking the right balance between greediness and maintaining diversity of the population. Greediness improves the quality of the individuals substantially, but overdoing it is detrimental to the overall performance of the genetic algorithm. We obtained the right balance through extensive computational testing. We believe that the greedy principles obtained in this paper might lead to improved genetic algorithms for other combinatorial optimization problems as well. It is plausible that using these ideas one could develop genetic algorithms for several hard combinatorial optimization problems that are better than or comparable to algorithms based on other heuristic optimization techniques.

## Acknowledgements

We thank Bezalel Gavish for suggesting us to investigate genetic algorithms for QAP. We also thank Fred Glover, Bettina Klinz, Abraham Punnen, Skorin-Kapov, Panos Pardalos and Mauricio Resende for their valuable help in this research by providing their feedback/papers/programs. Finally, we thank the two anonymous referees whose perceptive comments helped to improve the presentation substantially. This research was partially supported by ONR Grant N00014-94-1-0099 and a grant from UPS Foundation.

## References

[1] Burkard RE. Location with spatial interactions: the quadratic assignment problem. In: Mirchandani PB, Francis RL (Eds.), Discrete Location Theory, New York: Wiley, 1991.
[2] Malucelli F. Quadratic assignment problems: solution methods and applications. Unpublished Doctoral Dissertation, Departimento di Informatica, Universita di Pisa, Italy, 1993.
[3] Pardalos PM, Rendl F, Wolkowicz H. The quadratic assignment problem. In: Pardalos PM, Wolkowicz H (Eds.), Quadratic Assignment and Related Problems, DIMACS Series. Providence. RI: American Mathematical Society, 1994, pp. 1–42.
[4] Christofides N, Benavent E. An exact algorithm for the quadratic assignment problem. Oper. Res. 1989;37:760–8.
[5] Bazara MS, Sherali MD. Benders' partitioning scheme applied to a new formulation of the quadratic assignment problem. Naval Res. Logist. Quart. 1980;27:29–41.
[6] Lawler EL. The quadratic assignment problem. Manag. Sci. 1963;9:586–99.
[7] Pardalos PM, Crouse J. A parallel algorithm for the quadratic assignment problem. Proceedings of the Supercomputing 1989 Conference, ACM Press. New York, 1989, pp. 351–60.
[8] Armour GC, Buffa ES. Heuristic algorithm and simulation approach to relative location of facilities. Manag. Sci. 1963;9:294–309.
[9] Buffa ES, Armour GC, Vollmann TE. Allocating facilities with CRAFT. Harvard Business Rev. 1962;42:136–58.
[10] West DH. Algorithm 608: approximate solution of the quadratic assignment problem. ACM Trans. Math. Software 1983;9:461–6.
[11] Burkard RE, Bonniger T. A heuristic for quadratic boolean programs with applications to quadratic assignment problems. European J. Oper. Res. 1983;13:374–86.
[12] Li T, Pardalos PM, Resende MGC. A greedy randomized adaptive search procedure for the quadratic assignment problem. In: Pardalos PM, Wolkowicz H (Eds.), Quadratic Assignment and Related Problems. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Providence. RI: Americal Mathematical Society, 1994, pp. 237–261.
[13] Wilhelm MR, Ward TL. Solving quadratic assignment problems by simulated annealing. IEEE Trans. 1987;19:107–19.
[14] Skorin-Kapov J. Tabu search applied to the quadratic assignment problem. ORSA J. Comput. 1990;2:33–45.

[15] Taillard E. Robust tabu search for the quadratic assignment problem. Parallel Comput. 1991;17:443–55.
[16] Fleurent C, Ferland JA. Genetic hybrids for the quadratic assignment problem. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 16, Providence. RI: American Mathematical Society, 1994, pp. 173–87.
[17] Resende MGC, Pardalos PM, Li Y. Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP. ACM Trans. Math. Software 1994;22:104–18.
[18] Goldberg DE. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company, Reading, MA, 1989.
[19] Davis L. Handbook of Genetic Algorithms, Van Nostrand. New York, 1991.
[20] Burkard RE, Karisch SE, Rendl F. QAPLIB – A quadratic assignment program library. J. Global Optim. 1997;10:391–403.
[21] Bean JC. Genetic algorithms and random keys for sequencing and optimization. ORSA J. Comput. 1994;6:154–60.
[22] Tate DE, Smith AE. A genetic approach to the quadratic assignment problem. Comput. Oper. Res. 1985;22:73–83.
[23] Nugent CE, Vollman TE, Ruml J. An experimental comparison of techniques for the assignment of facilities to locations. Oper. Res. 1968;16:150–73.
[24] Holland JH. Adaptations in Natural and Artificial Systems. Ann Arbor: The University of Michigan Press, 1975.
[25] Glover F. Genetic algorithms and scatter search: unsuspected potential. Statist. Comput. 1994;4:131–40.
[26] Aggarwal CC, Orlin JB, Tai RP. Optimized crossover for the independent set problem. Oper. Res. 1997;45:226–34.

**Ravindra K. Ahuja** is a Professor in the Industrial & Systems Engineering Department at the University of Florida. In the past, he has held faculty positions at the Indian Institute of Technology, Kanpur, Massachusetts Institute of Technology, Cambridge, and Rutgers Univeristy, New Brunswick. His current research interests include network flows, routing and scheduling, airline scheduling, and logistic problems in supply-chain management.

**James B. Orlin** is the Edward Pennell Brooks Professor of Operations Reserach at the MIT Sloan School of Management, and is also a co-director of the MIT Operations Reserach Center. He specializes in developing efficient algorithms in network and combinatorial optimization. Together with Ravindra K. Ahuja and Thomas L. Magnanti, he co-authored the book, "*Network Flows*: *Theory*, *Algorithms*, *and Applications*", which won the 1993 Lanchester Prize.

**Ashish Tiwari** is a doctoral student in the Computer Science Department at the State University of New York at Stony Brook. He completed his B.S. at the Indian Institute of Technology in 1995. The research reported in this paper is based on his B.S. Project. His recent research work has focussed around term rewriting, automated deduction and computer algebra.