DIMACS Series in Discrete Mathematics
and Theoretical Computer Science

A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem

Yong Li, Panos M. Pardalos, and Mauricio G.C. Resende

ABSTRACT. A greedy randomized adaptive search procedure (GRASP) is a randomized heuristic that has been shown to quickly produce good quality solutions for a wide variety of combinatorial optimization problems. In this paper, we describe a GRASP for the quadratic assignment problem. We review basic concepts of GRASP: construction and local search algorithms. The implementation of GRASP for the quadratic assignment problem is described in detail. Computational experience on a large set of standard test problems (QAPLIB) is presented.

1. Introduction

Given a set $\mathcal{N} = \{1,2,\dots,n\}$ and $n \times n$ matrices $F = (f_{ij})$ and $D = (d_{kl})$, the quadratic assignment problem (QAP) can be stated as follows:

$$\min_{p \in \Pi_{\mathcal{N}}} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{p(i)p(j)},$$

where $\Pi_{\mathcal{N}}$ is the set of all permutations of $\mathcal{N}$. One of the major applications of the QAP is in location theory where the matrix $F = (f_{ij})$ is the flow matrix, i.e. f_{ij} is the flow of materials from facility i to facility j, and $D = (d_{kl})$ is the distance matrix, i.e. d_{kl} represents the distance from location k to location l [**9, 10, 21**]. The cost of simultaneously assigning facility i to location k and facility j to location l is $f_{ij}d_{kl}$. The objective is to find an assignment of all facilities to all locations (i.e. a permutation $p \in \Pi_{\mathcal{N}}$), such that the total cost of the assignment is minimized. Throughout this paper we often refer to the QAP in the context of this location problem.

In addition to its application in facility location problems, the QAP has been found useful in such applications as scheduling [**16**], the backboard wiring problem in electronics [**40**], and statistical data analysis [**17**]. Other applications may be found in [**15, 22, 28**].

The QAP is, computationally, one of the most difficult combinatorial optimization problems. This problem, of which the traveling salesman problem, graph isomorphism,

1991 *Mathematics Subject Classification.* Primary 90B80, 90C20, 90C35, 90C27; Secondary 65H20, 65K05.

Key words and phrases. Combinatorial optimization, quadratic assignment problem, local search, PLS-complete, GRASP, computer implementation, graph partitioning.

graph partitioning, and the band-width reduction problem are special cases [**4, 24**], is NP-hard. Moreover, unless P=NP, there exists no polynomial-time algorithm to find an ε-approximate solution [**34**] to the QAP. Furthermore, the quadratic assignment problem is PLS-Complete with respect to a Kernighan-Lin like neighborhood [**29**]. Computationally, problems of size $n > 20$ are not, to this date, practically solvable to optimality [**26, 27, 31, 33**]. Because of its complexity, many heuristic methods have been developed to solve the QAP [**3, 38, 39, 41**].

In this paper, we present a greedy randomized adaptive search procedure (GRASP) for the QAP. A GRASP is an iterative process, with each GRASP iteration consisting of two phases, a construction phase and a local search phase. The best overall solution is kept as the result.

In the first phase (construction phase), a feasible solution is iteratively constructed, one element at a time. At each construction iteration, the choice of the next element to be added is determined by ordering all elements in a candidate list with respect to a greedy function. This function measures the (myopic) benefit of selecting each element. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but usually not the top candidate. This choice technique allows for different solutions to be obtained at each GRASP iteration, but does not necessarily compromise the power of the adaptive greedy component of the method.

As is the case for many deterministic methods, the solutions generated by a GRASP construction are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. Normally, a local optimization procedure such as a two-exchange is employed. While such procedures can require exponential time from an arbitrary starting point, empirically their efficiency significantly improves as the initial solutions improve. Through the use of customized data structures and careful implementation, an efficient construction phase can be created which produces good initial solutions for efficient local search. The result is that often many GRASP solutions are generated in the same amount of time required for the local optimization procedure to converge from a single random start. Furthermore, the best of these GRASP solutions is generally significantly better than the solution obtained from a random starting point.

Figure 1 illustrates a generic GRASP implementation in pseudo-code. The GRASP takes as input parameters for setting the candidate list size, maximum number of GRASP iterations and the seed for the random number generator. After inputting the instance data (line 1) and initializing data structures (line 2) and the value of the best solution found (line 3), the GRASP iterations are carried out in lines 4–8. Each GRASP iteration consists of the construction phase (line 5), the local search phase (line 6) and, if necessary, the incumbent solution update (line 7).

An especially appealing characteristic of GRASP is the ease with which it can be implemented. First, adaptive greedy functions are known for many problems. Furthermore, neighborhood definitions and local search techniques are plentiful. Few parameters need to be set and tuned (candidate list size and number of GRASP iterations) and therefore development can focus on implementing efficient data structures to assure quick GRASP iterations. Finally, GRASP can be trivially implemented on a parallel processor in an MIMD environment. Each processor can be initialized with its own copy of the procedure, the instance data, and an independent random number sequence. The GRASP iterations

```
procedure grasp(ListSize,MaxIter,RandomSeed)
1   InputInstance();
2   InicializeDataStructures();
3   BestSolutionFound = ∞;
4   do k = 1,...,MaxIter →
5       ConstructGreedyRandomizedSolution(ListSize,RandomSeed);
6       LocalSearch(BestSolutionFound);
7       UpdateSolution(BestSolutionFound);
8   od;
9   return(BestSolutionFound)
end grasp;
```

FIGURE 1. A generic GRASP pseudo-code

are then performed in parallel with only a single global variable required to store the best solution found over all processors.

GRASP has been applied successfully to several combinatorial optimization problems. These include set covering problems arising from the incidence matrix of Steiner triple systems [**13**], maximum independent set problem [**14**], corporate acquisition of flexible manufacturing equipment [**2**], computer aided process planning [**1**], airline flight scheduling and maintenance base planning [**12**], scheduling of parallel machines [**23**], p-hub location problems [**19**], and component grouping [**20**].

The paper is organized as follows. In Section 2, we describe, in detail, the GRASP for QAP, including the construction and local search phases. Computational results on the QAPLIB [**5**] and QAP test problems with known optimal solutions [**26**] are given in Section 3. Concluding remarks are made in Section 4.

2. A GRASP for QAP

As outlined in Section 1, a GRASP possesses four basic components: a greedy function, an adaptive search strategy, a probabilistic selection procedure, and a local search technique. These components are linked together into an iterative method that constructs a feasible solution one element at a time and then feeds the solution to the local search procedure. When applied to the QAP, the permutations are formed in two phases: (*i*) the first 2 assignments, and (*ii*) the remaining $n-2$ assignments. The first two assignments are first made simultaneously in the first construction iteration, while the remaining $n-2$ are made one assignment per construction iteration. The greedy function chosen in this implementation orders admissible assignments with respect to cost. For the initial two assignments, the greedy choice is the pair of assignments with the minimum cost of interaction. The greedy function implemented in this GRASP assigns facilities with high interflow to nearby locations. The final $n-2$ assignments use as the greedy choice the assignment that has minimum cost interaction with respect to the already-made assignments. The term *admissible* initially refers to all couples of facility-location pairs and for the remaining $n-2$ assignments to facility-location pairs that have not yet been assigned. The local search implemented in this GRASP is a two-exchange heuristic. We next describe the GRASP components in detail.

2.1. Stage 1 of Construction Phase.

In Stage 1 of the construction phase, we select the pair of facilities and their matching locations, corresponding to the first 2 assignments of the constructed solution. One approach for making a greedy choice would be to sort all $O(n^4)$ $f_{ij}d_{kl}$ entries and select the couple of assignments having the smallest cost. Instead, we compromise in order to speed up the initialization process.

Let $0 < \beta < 1$ be a given candidate restriction parameter and $\lfloor x \rfloor$ be the largest integer smaller or equal to x. We sort the $n^2 - n$ distance entries in D, keeping the $\lfloor \beta(n^2-n) \rfloor$ smallest, i.e.

$$d_{i_1 j_1} \leq d_{i_2 j_2} \leq \cdots \leq d_{i_{\lfloor \beta(n^2-n) \rfloor} j_{\lfloor \beta(n^2-n) \rfloor}}$$

and sort the $n^2 - n$ flow entries in F, keeping the $\lfloor \beta(n^2-n) \rfloor$ largest, i.e.

$$f_{k_1 l_1} \geq f_{k_2 l_2} \geq \cdots \geq f_{k_{\lfloor \beta(n^2-n) \rfloor} l_{\lfloor \beta(n^2-n) \rfloor}}.$$

Then, the costs of interaction

$$d_{i_1 j_1} f_{k_1 l_1}, d_{i_2 j_2} f_{k_2 l_2}, \ldots, d_{i_{\lfloor \beta(n^2-n) \rfloor} j_{\lfloor \beta(n^2-n) \rfloor}} f_{k_{\lfloor \beta(n^2-n) \rfloor} l_{\lfloor \beta(n^2-n) \rfloor}}$$

are sorted in increasing order, keeping the smallest $\lfloor \alpha\beta(n^2-n) \rfloor$ elements, where α ($0 < \alpha < 1$) is the second candidate restriction parameter.

Note that the above ordering of the cost elements needs to be done only once, in the initialization phase of the GRASP. Since the data is static throughout the GRASP iterations, the order of the cost elements remains unchanged. This can be done efficiently with a heap sort.

In the GRASP iterations, the *greedy function* selects the couple of assignments having the smallest $d_{ij}f_{kl}$ from the $\lfloor \alpha\beta(n^2-n) \rfloor$ possible pairs. The *candidate restriction* limits our choice to those couples of assignment pairs having the $\lfloor \alpha\beta(n^2-n) \rfloor$ smallest $d_{ij}f_{kl}$ terms. The *random component* selects a couple of assignment pairs from the candidate list, at random.

Since the $\lfloor \alpha\beta(n^2-n) \rfloor$ cost elements are presorted, Stage 1 of the construction phase can be computed in constant time. All that is needed is to generate a random integer in the interval $[1, \lfloor \alpha\beta(n^2-n) \rfloor]$ and access the indices of the facilities and locations corresponding to the cost with the ranking given by the random number.

2.2. Stage 2 of Construction Phase. In Stage 2 of the construction phase of this GRASP, facilities are assigned to locations, one facility to one location at a time. Let

$$\Gamma = \{(j_1, l_1), (j_2, l_2), \ldots, (j_r, l_r)\}$$

be the set of already-made assignments. Stage 2 starts with $|\Gamma| = 2$, since in Stage 1 two pairs of assignments are made. Let

$$C_{ik} = \sum_{(j,l) \in \Gamma} f_{ij} d_{kl}$$

be the cost of assigning facility i to location k with respect to the already-made assignments.

The *greedy function* implemented in this GRASP is one that assigns a facility to a location that minimizes the total cost with respect to assignments already made, i.e. we select from the facility-location pairs not already assigned, the one that has the minimum C_{ik} cost.

Let there be, at a given GRASP iteration, m unassigned facility-location pairs and let α be the same candidate restriction parameter defined earlier. The *candidate restriction* used here limits our choice to the $\lfloor \alpha m \rfloor$ facility-location pairs having the smallest C_{ik} values. The *random component* of this GRASP selects at random from the candidate list a facility-location pair. The *adaptive component* is captured by updating the set Γ of already assigned

```
procedure stage2($\alpha$,$(j_1,l_1),(j_2,l_2)$)
1    $\Gamma = \{(j_1,l_1),(j_2,l_2)\}$;
2    do assignments $= 3,\dots,n \rightarrow$
3        $m = 0$;
4        do $i = 1,\dots,n \rightarrow$
5            do $k = 1,\dots,n \rightarrow$
6                if $(i,k) \notin \Gamma \rightarrow$
7                    $C_{ik} = \sum_{(j,l)\in\Gamma} f_{ij} d_{kl}$;
8                    inheap($C_{ik}$);
9                    $m = m + 1$;
10               fi;
11           od;
12       od;
13       $s =$ random$[1, \lfloor \alpha m \rfloor]$;
14       do $v = 1,\dots,s \rightarrow$
15           $C_{ik} =$ outheap();
16       od;
17       $\Gamma = \Gamma \cup \{i,k)\}$;
18   od;
19   return
end stage2;
```

FIGURE 2. Pseudo-code of Stage 2 of GRASP construction phase

pairs, i.e.

$$\Gamma = \Gamma \cup \{(i,k)\}.$$

In this fashion the greedy function changes at each GRASP iteration.

Figure 2 describes Stage 2 of the GRASP construction phase in pseudo-code. The procedure takes as input the initial two assignments, made in Stage 1 and the candidate list restriction parameter α. This procedure makes use of two heap operators: `inheap` inserts an element into the heap, updating the heap, and `outheap` deletes the top (smallest) element from the heap, and updates the heap. The set of assignments Γ initially consists of the two Stage 1 assignments (line 1). In lines 2–18 the remaining assignments are made. For each assignment, the candidate counter m is set in line 3 and in lines 4–12 the assignment costs of the unassigned pairs are computed and inserted in the heap structure for sorting. In line 13 a random number s is generated in the interval $[1, \lfloor \alpha m \rfloor]$ and in lines 14–16, the assignment pair having the s-th smallest cost is retrieved from the heap. Line 17 updates the solution set. We have omitted some trivial implementation details that make the above pseudo-code more efficient in order to make a cleared description of Stage 2.

2.3. Local Search. For a given problem, a local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution in the neighborhood of the current solution. It terminates when there is no better solution found in the neighborhood with respect to some cost function.

In this subsection, we discuss, with respect to the QAP, issues related to local search algorithms, neighborhood structures, techniques for searching the neighborhood of a solution, and related complexity issues.

```
procedure local($P$,$N(P)$,$s$)
1   do $s$ is not locally optimal $\rightarrow$
2       Find a better solution $t \in N(s)$;
3       Let $s = t$;
4   od;
5   return($s$ as local optimal for $P$)
end local;
```

FIGURE 3. Layout of Local Search

The *neighborhood structure* N for a problem P relates a solution s of the problem to a subset of solutions $N(s)$. A solution s is said to be *locally optimal* if there is no better solution in $N(s)$. Given a neighborhood structure N, a local search algorithm has the general form as stated in Figure 3.

The key to success for a local search algorithm consists of the suitable choice of a neighborhood structure, efficient neighborhood search techniques, and the starting solution. The GRASP construction phase plays an important role with respect to this last point, since it produces good starting solutions for local search. Next, the common neighborhood structures used in local search algorithms are discussed and a new neighborhood structure for the QAP is presented. Neighborhood design principles and neighborhood searching techniques are discussed.

Given two permutations p and q, the *difference* between p and q is defined to be $\delta(p,q) = \{i \mid p(i) \neq q(i)\}$, and the *distance* between p and q is defined to be $d(p,q) = |\delta(p,q)|$. To design a good neighborhood structure, one should be guided the following three principles: (a) reasonable neighborhood size; (b) large variance in neighborhood; and (c) high connectivity in neighborhood.

Principle (a) implies that one should limit the size of a neighborhood so that searching the neighborhood can be done efficiently. Define a *diameter* of a set of permutations S to be

$$D(S) = \max_{p,q \in S} d(p,q).$$

Principle (b) requires that the diameter of a neighborhood be large. Principle (c) implies that given two permutations in a neighborhood, there should be a sequence of permutations in the neighborhood such that the difference between each pair of consecutive permutations is small. For a given problem P with the solution space represented by $S(P)$, the *degree of connectivity* of a neighborhood structure N is defined to be

$$C_N = 1/K,$$

where

$$K = \min_{r \in S(P)} \min_{p,q \in N(r)} \max_{p_0 = p, p_k = q} \min_{i=1,\ldots,k-1} d(p_i, p_{i+1}).$$

The constant K can be interpreted as follows. First, for each pair of permutations $\{p,q\}$ in the neighborhood $N(r)$ of the same permutation r, there corresponds a smallest distance between consecutive permutations in a path in the neighborhood $N(r)$ between p and q. Taking the maximum among all such paths and then taking the minimum among all possible permutations in the set of solution space $S(P)$, one obtains K. Principle (c) requires that a good neighborhood have a high degree of connectivity. This principle rules out the use of random neighborhoods, for which the degree of connectivity is low.

```
procedure kexhange($k$,$N_k(p)$,$p$)
1   do $p$ is not locally optimal in $N_k(p)$ $\rightarrow$
2       Find a better solution $q \in N_k(p)$;
3       Let $p = q$;
4   od;
5   return($p$ as local optimal permutation)
end kexchange;
```

FIGURE 4. *k*-Exchange Neighborhood Local Search for the QAP

2.3.1. *Local Search Algorithms for the QAP.* In this subsection, algorithms for the QAP corresponding to the 3 types of neighborhood structures are presented. In a GRASP, a local search algorithm starts with an initial permutation produced by the construction phase and works in a iterative fashion. At each iteration of the local search, a better solution in the neighborhood of the current solution is pursued and the current solution is replaced by an improved solution, if one is found. The most commonly used neighborhood structure for the QAP is the so-called k-exchange neighborhood structure. The k-exchange neighborhood for a permutation $p \in \Pi_{\mathcal{N}}$ is defined to be

$$N_k(p) = \{q \mid d(p,q) \leq k\}, \text{ where } 2 \leq k \leq n.$$

The k-exchange neighborhood is also used for many other combinational optimization problems, such as the traveling salesman problem (TSP), the graph partitioning problem (GP) and the maximum clique problem (MCP). Pseudo-code for the k-exchange local search is shown in Figure 4.

Let p be the starting permutation of a local search algorithm and p^* the optimal permutation of the QAP and let k be the distance between p and p^*. If k is small, then a local search algorithm enumerating all permutations in $N(p,k)$ will yield the optimal solution. Among all possible values of k, the most popular is $k = 2$. This local search is used in the computational experiments of Section 3. When the value of k is large, searching the neighborhood can be too expensive. Since the TSP, GP, and MCP are special cases of the QAP, it is not surprising that the 2-exchange neighborhood structure for the QAP is a generalization of the 2-exchange neighborhood structures for the TSP, GP, and MCP.

The relationship between the GP and the QAP also gives rise to the adaptation of the λ-exchange neighborhood structure for the GP to the QAP. First, define the set of all possible pairwise exchanges as follows

$$E = \{(i,j) \mid i \neq j, i,j = 1,...,n\}.$$

For a given permutation p, denote by $N_2(p,E')$ the set of permutations in $N_2(p)$ obtained by performing pairwise exchanges in a subset of pairwise exchanges E'.

The λ-exchange neighborhood structure, denoted by N_λ, is defined as a union of a collection of subsets of 2-exchange neighborhoods. At each local search step, the current permutation is denoted as p_0 and the set $E_0 = E$. At step k of a search step, permutations $p_1,...,p_k$ have already been constructed. Let the best solution (tie is broken arbitrarily) in $N_2(p_k,E_k)$ be denoted p_{k+1}. Now, construct the set E_{k+1} from E_k as follows:

$$E_{k+1} = E_k - \{(i,j) \mid i \text{ or } j \in \bigcup_{l=0}^{k} \delta(p_l,p_{l+1}), i,j = 1,...,n\}. \quad (1)$$

```
procedure LambdaExchange($n$,$p$)
1   Loop $=$ true;
2   do Loop $==$ true $\rightarrow$
3       $p_0 = p$; $E_0 = E$; Done $=$ false;
4       do Done $==$ false and $E_k \neq \emptyset \rightarrow$
5           Find best solution $p_{k+1} \in N_2(p_k, E_k)$;
6           $E_{k+1} = E_k - \{(i,j) \mid i$ or $j \in \bigcup_{l=0}^{k} \delta(p_l, p_{l+1}), i,j = 1,...,n\}$;
7           if $f(p_{k+1}) \geq f(p_0) \rightarrow$ Done $=$ true fi;
8           $k = k+1$;
9       od;
10      $p = \mathrm{argmin}\{f(p_1), \ldots, f(p_k)\}$;
11      if $f(p) == f(p_0) \rightarrow$ Loop $=$ false fi;
12  od;
13  return(the local optimal $p$)
end LambdaExchange;
```

FIGURE 5. λ-Exchange Neighborhood Local Search

The process is continued until either $f(p_k) > f(p_0)$ or $E_k = \emptyset$. Then

$$N_\lambda(p_0) = \bigcup_{i=0}^{k} N_2(p_i, E_i).$$

The algorithm is stated in Figure 5.

Following the design principles of neighborhood structures, we propose a new neighborhood structure with the objective of providing better solutions than those obtained by the λ-exchange neighborhood. The new neighborhood structure, denoted by N^*, can also be defined as a union of a collection of subsets of 2-exchange neighborhoods. At each local search step, the permutation is denoted as p_0 and the set $E_0 = E$. At step k, permutations $p_1,...,p_k$ have already been constructed. Let the best solution (tie is broken arbitrarily) in $N_2(p_k, E_k)$ be denoted p_{k+1}. As before, construct the set E_{k+1} from E_k, this time according to

$$E_{k+1} = E_k - \{(i,j) \mid i \text{ and } j \in \bigcup_{l=0}^{k} \delta(p_l, p_{l+1}), i,j = 1,...,n\}. \tag{2}$$

The process is continued until either $f(p_k) > f(p_0)$ or $E_k = \emptyset$. Then

$$N^*(p_0) = \bigcup_{i=0}^{k} N_2(p_i, E_i).$$

The new local search algorithm corresponding to the N^* neighborhood structure is given in Figure 6.

Table 1 can be used to evaluate the neighborhood structures discussed earlier. The sizes of N_λ and N^* are approximate. Only the leading terms of the size are given. The exact formulae for the size of the neighborhood structures are given in Subsection 2.3.2.

An examination of Table 1 shows that N^* is slightly greater than N_λ. This is a disadvantage when one is more concerned about computational time. However, in general, N^* admits better solutions, as indicated by the computational results in [**25**]. For N_k with k small ($k \leq 4$), the neighborhood size is comparable with those of N_λ and N^*; however,

```
procedure NStarNeighborhood($n$,$p$)
1   Loop = true;
2   do Loop == true →
3       $p_0 = p$; $E_0 = E$; Done = false;
4       do Done == false and $E_k \neq \emptyset$ →
5           Find best solution $p_{k+1} \in N_2(p_k, E_k)$;
6           $E_{k+1} = E_k - \{(i,j) \mid i \text{ and } j \in \bigcup_{l=0}^{k} \delta(p_l, p_{l+1}), i,j = 1,...,n\}$;
7           if $f(p_{k+1}) \geq f(p_0)$ → Done = true fi;
8           $k = k+1$;
9       od;
10      $p = \text{argmin}\{f(p_1), \ldots, f(p_k)\}$;
11      if $f(p) == f(p_0)$ → Loop = false fi;
12  od;
13  return(the local optimal $p$)
end NStarNeighborhood;
```

FIGURE 6. N^* Neighborhood Local Search

TABLE 1. Theoretical Evaluation of Neighborhood Structures

	N_k	N_λ	N^*
Diameter	$\min\{2k, n\}$	n	n
Size	C_k^n	$n^3/24$	$n^4/8$
Connectivity	1/2	1/2	1/2

the diameter is smaller. For N_k with k large ($k > 4$), the neighborhood size is considerably larger than those of N_λ and N^*, while the diameter is comparable in size. Consequently, in general, N_k is not as good as N_λ and N^*.

2.3.2. *Searching the Neighborhoods.* The search for a better solution in a neighborhood $N(p)$ for a given solution s can be done in two ways: (i) *complete enumeration*, where one searches all the solutions in the neighborhood to find the best solution; (ii) *first decrement*, where one enumerates the solutions in the neighborhood and stop once a better solution is found. In both cases, the worst case time complexity of searching the neighborhood is equal to the size of the neighborhood. The size of the N_k neighborhood is $C_k^n = \frac{n!}{k!(n-k)!}$. The size S_λ of the N_λ neighborhood (assuming that $n = 2k$ for some k) is given by:

$$\begin{aligned} S_\lambda &= C_2^n + C_2^{n-2} + \ldots + C_2^2 \\ &= \sum_{i=1}^{k} \frac{2i(2i-1)}{2} \\ &= \sum_{i=1}^{k} (2i^2 - i) \\ &= \frac{1}{24}n^3 + \frac{5}{8}n^2 + \frac{7}{12}n \\ &= O(n^3). \end{aligned}$$

The size S^* of the new neighborhood N^* is given by:

$$\begin{aligned} S^* &= 1 + 2 + \ldots + C_2^n \\ &= \frac{1}{2}C_2^n(C_2^n - 1) \\ &= \frac{1}{2}\frac{n(n-1)}{2}(\frac{n(n-1)}{2} - 1) \\ &= \frac{1}{8}(n^4 - 2n^3 - n^2 + 2n) \\ &= O(n^4). \end{aligned}$$

Hence, searching the neighborhood N_k can be quite expensive when k is large while searching neighborhoods N_λ and N^* is more affordable.

2.3.3. *Complexity Issues.* Although local search algorithms work quite well in practice for many combinational optimization problems, they may require exponential time in worst case instances. In order to characterize the complexity of such local search algorithms, a new complexity class, the Polynomial-time Local Search (PLS) class, was introduced and studied in [**18**].

Let $I(P)$ denote the set of instances of the problem P with an associated cost function C. For an instance $x \in I(P)$, there exists a set of feasible solutions $F(x)$. For each feasible solution $s \in F(x)$, there exists a set of neighboring solutions $N(s,x)$.

Problem P is said to be in PLS if, given an input x and a set of feasible solutions $F(x)$, the following three polynomial-time algorithms exist:

1. On input $x \in I$, compute an initial feasible solution $s_0 \in F(x)$.
2. On input $x \in I$ and $s \in F(x)$ compute the corresponding cost $C(s,x)$.
3. On input $x \in I$ and $s \in F(x)$, either determine that s is locally optimal or find a better solution in $N(s,x)$, with respect to the cost function.

A problem $P \in$ PLS is PLS-reducible to another problem $Q \in$ PLS, if there are polynomial-time computable functions f_1 and f_2, such that f_1 maps an instance x of P to an instance $f_1(x)$ of Q and for any locally optimal solution s for $f_1(x)$, $f_2(s,x)$ produces a locally optimal solution for x. A problem $P \in$ PLS is PLS-complete, if every other problem in PLS is PLS-reducible to P.

The GP with 2-exchange neighborhood is PLS-complete [**29, 35**]. Since the GP is a special case of the QAP, the following theorem holds:

THEOREM 1. *The local search problems for the QAP with the k-exchange neighborhood structure, λ-exchange neighborhood structure, and the N^* neighborhood structure are PLS-complete.*

TABLE 2. GRASP runs on problem class BUR

Problem	Best value	GRASP	GRASP CPU time	
Name	found	iterations	local	total
bur26a	5426670	388	28.44	44.07
bur26b	3817852	81	5.67	8.90
bur26c	5426795	357	26.50	40.88
bur26d	3821225	158	11.20	17.79
bur26e	5386879	2179	164.80	755.98
bur26f	3782044	87	6.25	9.83
bur26g	10117172	1894	142.39	218.28
bur26h	7098658	283	21.03	32.44

TABLE 3. Permutations found by GRASP on problem class BUR

Problem Name	Permutation of best assignment
bur26a	13,8,17,5,12,9,4,15,11,20,1,7,6,16,2,23,21,10,19,18,14,25,26, 24,22,3
bur26b	12,25,17,5,13,7,4,15,11,20,1,16,26,6,3,24,22,10,19,18,14,9,8, 23,21,2
bur26c	17,2,1,15,19,21,24,12,10,8,6,3,4,18,9,5,25,13,11,14,16,20,23, 22,7,26
bur26d	18,4,2,23,19,21,24,7,10,25,22,11,1,17,9,5,26,13,12,14,8,20,15, 3,16,6
bur26e	9,23,15,1,19,20,4,16,18,21,25,13,2,3,10,5,8,12,14,11,17,24,26, 22,7,6
bur26f	9,21,22,24,19,2,3,25,18,8,26,14,1,15,10,5,4,11,13,12,17,23,6, 20,16,7
bur26g	9,1,24,12,19,20,11,8,17,25,2,15,5,23,16,21,26,14,18,13,10,3,4, 6,7,22
bur26h	9,2,22,11,19,5,13,8,17,6,1,4,3,15,16,20,26,14,18,12,10,21,23, 25,7,24

3. Experimental Results

In this section, we report experimental results describing the testing of a FORTRAN implementation of the GRASP described in this paper on a wide range of test problems, including most instances from the suite of QAP test problem QAPLIB [**5**] and a new class of test problems with known optimal solutions [**26**]. The objective of this experiment is to show the effectiveness of a simple GRASP code in obtaining good-quality solutions quickly.

The FORTRAN code implementation has 537 lines of code, including input/output and debugging statements and was written and debugged in a few hours. We used the portable random number generator `rand` of Schrage [**36**] with the initial seed 270001 to generate the random integer to select a candidate from the restricted candidate list.

The experiment was conducted on a single processor of a Silicon Graphics Challenge computer (100 MHz MIPS R4400 processor), using the SGI FORTRAN compiler `f77` with

TABLE 4. GRASP runs on problem class CHR

Problem Name	Best value found	GRASP iterations	GRASP CPU time local	GRASP CPU time total
chr12a	9552	39	0.15	0.26
chr12b	9742	12	0.06	0.09
chr12c	11156	321	1.56	2.16
chr15a	9896	13475	128.11	197.76
chr15b	7990	1118	10.54	16.32
chr15c	9504	6484	60.76	93.79
chr18a	11098	2479	42.09	67.86
chr18b	1534	58	0.93	1.56
chr20a	2192	94307	2328.91	3785.86
chr20b	2370	61232	1442.65	2388.01
chr20c	14142	1461	30.55	53.18
chr22a	6190	37724	1336.41	2175.09
chr22b	6282	238	8.46	13.81
chr25a	3972	15565	845.62	1412.09

TABLE 5. Permutations found by GRASP on problem class CHR

Problem Name	Permutation of best assignment
chr12a	5,4,6,12,2,10,1,11,7,9,8,3
chr12b	3,8,6,7,1,10,2,12,9,4,5,11
chr12c	3,11,4,6,2,8,1,7,9,5,10,12
chr15a	14,8,13,9,1,10,11,3,15,2,6,5,4,7,12
chr15b	4,6,12,1,7,9,11,15,5,13,14,8,2,10,3
chr15c	6,2,10,9,3,8,4,5,12,15,14,13,1,7,11
chr18a	9,18,1,4,8,3,12,11,15,7,10,6,2,14,17,16,13,5
chr18b	3,5,7,1,18,9,2,16,11,4,14,13,6,12,15,8,10,17
chr20a	11,15,1,8,16,12,3,14,5,9,10,6,19,17,13,18,20,4,7,2
chr20b	8,14,4,18,13,3,2,9,10,11,12,1,16,5,20,7,15,17,19,6
chr20c	18,4,7,8,14,2,11,19,3,5,6,1,12,15,9,13,16,10,17,20
chr22a	6,2,15,16,11,13,7,4,19,21,14,22,10,20,1,5,9,8,18,17, 3,12
chr22b	4,6,11,10,17,2,9,1,18,8,13,19,14,16,3,21,7,22,20,15, 12,5
chr25a	6,11,4,18,3,12,14,20,13,10,8,22,17,23,24,7,21,5,16,9, 19,2,25,15,1

TABLE 6. GRASP run on problem class ELS

Problem Name	Best value found	GRASP iterations	GRASP CPU time local	GRASP CPU time total
els19.out	17212548	105	2.63	3.99

TABLE 7. Permutation found by GRASP on problem class ELS

Problem Name	Permutation of best assignment
els19	17,18,19,11,12,9,3,14,1,2,10,13,7,5,15,16,8,4,6

TABLE 8. GRASP runs on problem class ESC

Problem	Best value	GRASP	GRASP CPU time	
Name	found	iterations	local	total
esc08a	2	1	0.00	0.00
esc08b	8	1	0.00	0.00
esc08c	32	2	0.00	0.00
esc08d	6	1	0.00	0.00
esc08e	0	0	0.00	0.00
esc08f	18	2	0.00	0.00
esc16a	68	1	0.01	0.02
esc16b	292	1	0.01	0.02
esc16c	160	2	0.01	0.03
esc16d	16	1	0.01	0.02
esc16e	28	12	0.10	0.17
esc16f	0	0	0.00	0.00
esc16g	26	1	0.01	0.02
esc16h	996	1	0.01	0.02
esc16i	14	1	0.01	0.02
esc16j	8	1	0.01	0.02
esc32a	130	22339	2256.88	4780.14
esc32b	168	94	10.08	18.91
esc32c	642	2	0.18	0.37
esc32d	200	22	1.84	3.92
esc32e	2	1	0.06	0.16
esc32f	2	1	0.06	0.16
esc32g	6	1	0.06	0.16
esc32h	438	28	2.66	5.26
esc64a	116	1	0.96	2.44
esc128	64	21	139.79	580.21

the compiler flags `-O2 -Olimit 800`. CPU times in seconds were computed by calling the system routine `etime()`. Reported CPU times exclude problem input time.

GRASP requires few parameters to be set. The performance of most heuristics depends on parameter setting. Since we would like to make our results as reproducible as possible, we limit our runs in this experiment to a single set of parameter settings. Throughout the experiment we used the following parameters:

- $\alpha = 0.5$
- $\beta = 0.1$
- `maxiter` $= 100,000$.

By using the same parameter setting we also illustrate the robustness of this approach.

TABLE 9. Permutations found by GRASP on problem class ESC

Problem Name	Permutation of best assignment
esc08a	1,2,8,5,7,6,3,4
esc08b	4,3,8,2,6,1,7,5
esc08c	2,6,8,5,4,7,1,3
esc08d	3,7,5,6,1,4,2,8
esc08e	7,8,4,1,2,6,3,5
esc08f	8,5,7,6,1,2,3,4
esc16a	9,3,12,11,10,2,5,16,8,14,6,13,1,4,7,15
esc16b	5,8,13,12,11,3,14,1,4,7,10,15,6,2,9,16
esc16c	12,14,13,16,2,3,4,5,11,1,9,15,10,7,6,8
esc16d	14,12,6,15,8,13,5,3,11,9,7,2,10,16,4,1
esc16e	3,16,1,4,15,13,6,9,14,11,7,2,10,12,8,5
esc16f	14,16,15,12,13,3,2,4,11,8,10,6,1,9,7,5
esc16g	6,1,5,9,4,11,2,16,7,15,13,12,8,10,3,14
esc16h	7,6,14,13,9,10,15,12,4,8,16,3,11,5,1,2
esc16i	6,7,1,14,5,13,16,10,3,8,2,9,4,11,12,15
esc16j	13,11,1,14,6,4,10,16,7,15,8,9,5,2,12,3
esc32a	5,2,4,3,9,14,18,15,6,1,32,26,11,13,19,17,29,31,23,16,28, 30,21,8,7,22,10,12,20,25,24,27
esc32b	24,22,23,21,18,12,17,11,25,26,20,19,29,28,6,5,15,9,13,7, 16,10,14,8,31,30,3,1,27,32,4,2
esc32c	14,6,23,22,16,13,28,21,17,18,27,24,10,19,2,31,5,8,32,30, 11,12,25,4,9,20,26,3,15,7,29,1
esc32d	18,17,7,8,1,10,5,14,19,28,22,27,4,13,3,12,16,6,9,15,11,24, 2,21,23,20,29,30,32,25,31,26
esc32e	7,13,8,12,9,32,17,20,19,30,21,3,16,10,6,18,5,14,2,1,31,24, 23,27,11,29,4,28,15,22,25,26
esc32f	7,13,8,12,9,32,17,20,19,30,21,3,16,10,6,18,5,14,2,1,31,24, 23,27,11,29,4,28,15,22,25,26
esc32g	4,18,19,32,3,30,28,14,26,11,31,29,16,10,12,1,5,7,21,17,2,24, 25,27,22,13,8,9,15,23,6,20
esc32h	13,14,7,3,22,18,5,31,30,25,24,8,28,27,20,12,17,16,4,15,6,10, 23,2,26,1,19,9,29,32,21,11
esc64a	4,9,20,33,62,24,27,35,10,15,44,14,46,11,31,22,50,7,47,21,36, 5,49,16,8,37,6,19,17,57,41,58,40,42,25,30,51,43,39,3,59,54,2, 26,55,12,45,52,38,32,23,28,29,64,48,34,18,13,61,60,63,56,53,1
esc128	80,75,66,79,71,73,77,69,20,6,49,25,115,81,122,119,85, 91,32,28,48,87,9,5,123,126,16,26,47,94,120,92,43,2,11, 72,12,27,128,46,62,57,21,86,30,58,98,10,125,78,116,45, 84,70,41,54,106,99,38,18,52,34,7,105,36,110,35,63,31, 90,33,111,55,60,96,23,83,117,8,82,67,29,114,97,51,108, 76,113,101,124,14,65,53,88,107,59,40,44,127,37,42,24, 112,104,64,74,15,17,95,19,50,13,4,102,103,39,56,68,109, 118,100,121,1,93,3,61,89,22

TABLE 10. GRASP runs on problem class KRA

Problem	Best value	GRASP	GRASP CPU time	
Name	found	iterations	local	total
kra30a	88900	16786	1738.07	2919.50
kra30b	91420	51537	5364.87	8988.54

TABLE 11. Permutations found by GRASP on problem class KRA

Problem Name	Permutation of best assignment
kra30a	26,24,19,16,20,23,6,10,11,2,22,18,7,14,15,21,25,29,12,9,5,17,1, 8,13,28,30,3,4,27
kra30b	23,22,25,19,20,26,5,8,9,2,21,18,6,12,16,24,27,30,13,7,4,17,1,11, 14,29,15,3,10,28

TABLE 12. GRASP runs on problem class NUG

Problem	Best value	GRASP	GRASP CPU time	
Name	found	iterations	local	total
nug05	52	1	0.00	0.00
nug06	86	1	0.00	0.00
nug07	148	1	0.00	0.00
nug08	214	4	0.00	0.01
nug12	578	100	0.47	0.64
nug15	1150	20	0.17	0.27
nug20	2570	736	19.39	30.12
nug30	6124	79861	8889.22	14406.48

TABLE 13. Permutations found by GRASP on problem class NUG

Problem Name	Permutation of best assignment
nug05	4,3,5,2,1
nug06	3,2,1,6,5,4
nug07	1,2,5,3,4,7,6
nug08	6,5,1,7,8,4,3,2
nug12	5,1,9,8,4,3,11,7,10,2,6,12
nug15	11,12,7,6,4,3,9,14,15,1,10,5,13,8,2
nug20	19,7,4,6,17,20,18,14,5,3,9,8,15,2,12,10,16,1,11,13
nug30	14,29,4,12,25,27,16,15,22,21,9,26,28,5,1,10,13,2,17, 6,30,8,7,20,18,19,3,24,23,11

We tested the GRASP implementation on most of the problem classes in QAPLIB: BUR [**6**], CHR [**8**], ELS [**10**], ESC [**11**], KRA [**22**], NUG [**30**], ROU [**32**], SCR [**37**], SKO [**39**], and STE [**40**]. The only class left out was CAR [**7**] which has QAP instances with a nonzero linear part and cannot be handled by our current GRASP implementation. We

TABLE 14. GRASP runs on problem class ROU

Problem	Best value	GRASP	GRASP CPU time	
Name	found	iterations	local	total
rou10	174220	34	0.06	0.12
rou12	235528	307	1.48	2.02
rou15	354210	4	0.05	0.06
rou20	725522	31246	853.36	1308.09

TABLE 15. Permutations found by GRASP on problem class ROU

Problem Name	Permutation of best assignment
rou10	9,4,1,10,8,5,6,3,7,2
rou12	8,5,7,11,2,1,10,6,4,12,3,9
rou15	10,8,6,13,5,2,9,3,11,12,15,1,4,14,7
rou20	1,3,16,12,10,20,11,13,9,5,7,17,19,4,15,6,18,14,2,8

TABLE 16. GRASP runs on problem class SCR

Problem	Best value	GRASP	GRASP CPU time	
Name	found	iterations	local	total
scr10	26992	6	0.00	0.02
scr12	31410	28	0.15	0.20
scr15	51140	30	0.34	0.48
scr20	110030	325	9.64	14.47

TABLE 17. Permutations found by GRASP on problem class SCR

Problem Name	Permutation of best assignment
scr10	6,4,3,9,7,2,10,1,8,5
scr12	11,12,5,8,1,10,2,7,9,3,4,6
scr15	5,8,7,6,12,10,2,4,15,14,3,9,11,13,1
scr20	20,8,7,5,18,4,2,6,12,19,10,3,17,9,14,15,16,11,13,1

also tested the GRASP on two new classes of QAP test problems, with known optimal solutions, generated with the generator described in [**26**]: LIPAA and LIPAB.

For each problem class we report the value of the best assignment found, GRASP iterations and total and local search CPU time to find the first occurrence of the best assignment, and the permutation of the first best solution found.

Tables 2–3 summarize results for problem class BUR. In all cases the GRASP found solutions having better objective value than those reported in the literature. Tables 4–5 show computational results for class CHR. In all, but four instances, the GRASP produced optimal permutations. Tables 6–7 show results for the single ELS instance, where the GRASP produced an optimal permutation in less than four seconds. Testing of problem class ESC is summarized in Tables 8-9. Most of those instances were solved in fractions

TABLE 18. GRASP runs on problem class STE

Problem	Best value	GRASP	GRASP CPU time	
Name	found	iterations	local	total
ste36a	9588	29836	6351.89	10590.64
ste36b	15852	9640	2228.62	3600.14
ste36c	8254628	408	92.57	150.65

TABLE 19. Permutations found by GRASP on problem class STE

Problem Name	Permutation of best assignment
ste36a	16,18,35,25,34,33,15,26,27,17,32,23,24,5,6,7,9,8,4,14,21,30,22, 28,29,20,31,13,12,11,10,3,1,2,19,36
ste36b	31,10,2,12,3,4,22,11,29,21,5,14,13,15,30,28,19,20,24,23,7,8,16, 9,17,18,6,32,33,34,35,25,27,26,36,1
ste36c	31,10,2,12,3,4,22,11,30,21,5,14,13,15,32,29,19,20,25,23,17,18,16, 9,8,7,6,24,33,34,35,26,27,36,1,28

TABLE 20. GRASP runs on problem class LIPAA

Problem	Optimal	Best value	GRASP	GRASP CPU time	
Name	value	found	iterations	local	total
lipa10a	473	473	4	0.01	0.01
lipa20a	3683	3683	35	0.86	2.18
lipa30a	13178	13178	703	71.11	119.58
lipa40a	31538	31538	97667	27062.88	47390.47
lipa50a	62093	62655	67360	41600.45	80382.47
lipa60a	107218	108118	84179	93574.86	189669.64
lipa70a	169755	171021	67093	128319.19	266824.13
lipa80a	253195	254907	77868	236454.84	505766.13
lipa90a	360630	362847	31640	152307.84	328593.53

of a second in one GRASP iteration. One instance, esc32a, appears to be much harder than the others. In all cases, GRASP produced the best known solution and for problem esc128, a solution of cost 64 was found (in [**5**] a best known value of 84 is reported). Tables 10–11 give results for problem class KRA. For those two instances, GRASP produced best known solutions, requiring over two hours of CPU time for kra30b. Results for the classical problem set NUG are given in Tables 12–13. The GRASP found best known solutions for all of the instances. In most cases we found optimal permutations that were different from those reported in [**5**]. Tables 14–15 show results for problem class ROU, where the GRASP produced best known solutions for all four instances. Testing on problem class SCR is summarized in Tables 16–17. GRASP produced optimal permutations for all four instances. Tables 18–19 summarize test results for the three STE instances. In only one of the three instances, did the GRASP produce the best known solution (ste36b). For the ste36c instance, GRASP found a solution within 0.2% of the best known solution in less than 151 CPU seconds.

TABLE 21. Permutations found by GRASP on problem class LIPAA

Problem Name	Permutation of best assignment
lipa10a	4,3,10,1,6,9,8,5,2,7
lipa20a	4,19,13,9,5,12,3,18,6,7,15,8,17,14,16,10,2,20,1,11
lipa30a	28,30,26,16,11,10,22,18,1,29,9,8,2,25,23,27,4,17,19, 13,21,3,6,24,5,15,14,12,7,20
lipa40a	10,8,24,25,15,2,1,18,31,28,26,40,29,3,36,9,32,34,5,19, 7,17,14,12,37,22,4,16,23,13,33,21,35,38,20,27,6,30,39,11
lipa50a	32,34,11,33,7,2,25,31,29,49,50,40,26,30,47,28,17,5,10, 16,38,21,6,39,20,18,46,1,27,12,36,45,14,23,24,44,8,9,4, 41,22,43,37,3,13,42,48,19,15,35
lipa60a	21,1,8,6,49,44,13,43,32,53,58,54,45,60,16,4,14,5,37,51, 50,35,56,30,18,59,15,25,48,33,2,34,12,36,24,41,19,52,3,38, 40,22,46,7,23,17,39,31,57,9,10,20,28,11,42,27,47,29,26,55
lipa70a	65,21,68,12,7,44,17,61,27,70,14,55,67,31,43,25,32,48,22,58, 52,57,29,63,11,28,66,6,24,35,45,62,50,30,34,2,40,42,13,9,19, 49,47,56,53,10,1,59,64,46,69,26,36,16,39,18,41,5,38,23,20,8, 15,3,51,4,54,60,37,33
lipa80a	66,14,50,27,43,42,36,78,18,68,26,69,56,16,65,64,79,53,72,35, 5,11,13,54,51,32,31,22,47,76,67,38,44,1,10,45,63,6,30,4,8,41, 7,59,80,20,37,62,29,15,34,3,23,28,74,17,40,61,33,52,48,25,57, 70,73,55,21,39,12,75,58,77,24,49,9,46,19,60,2,71
lipa90a	53,27,35,26,2,86,82,67,88,89,69,12,24,40,5,75,8,3,19,29,47,80, 15,70,11,84,16,20,31,28,58,66,71,32,25,56,61,1,14,33,65,83,59, 6,43,76,30,55,37,48,9,4,23,13,74,7,68,63,62,45,85,18,77,46,54, 60,39,72,50,21,38,41,22,17,44,34,64,78,73,52,79,51,87,36,90,49, 10,57,81,42

TABLE 22. GRASP runs on problem class LIPAB

Problem	Optimal	Best value	GRASP	GRASP CPU time	
Name	value	found	iterations	local	total
lipa10b	2008	2008	1	0.00	0.00
lipa20b	27076	27076	16	0.47	0.71
lipa30b	151426	151426	27	3.19	5.03
lipa40b	476581	476581	43	14.12	23.09
lipa50b	1210244	1210244	212	154.64	258.62
lipa60b	2520135	2520135	384	505.04	886.46
lipa70b	4603200	4603200	81	191.66	338.82
lipa80b	7763962	7763962	592	2179.96	4003.01
lipa90b	12490441	12490441	4462	25023.39	47141.35

Tables 20–21 show results for the first new problem class LIPAA, and tables 22–23 show results for the other new problem class LIPAB. In addition to the information provided in the tables for the other problem classes, tables 20 and 22 also display the known optimal value for each instance. On problem class LIPAA, the GRASP found an optimal

TABLE 23. Permutations found by GRASP on problem class LIPAB

Problem Name	Permutation of best assignment
lipa10b	1,2,3,4,5,6,7,8,9,10
lipa20b	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
lipa30b	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20, 21,22,23,24,25,26,27,28,29,30
lipa40b	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21, 22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40
lipa50b	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22, 23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41, 42,43,44,45,46,47,48,49,50
lipa60b	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22, 23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41, 42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60
lipa70b	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22, 23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41, 42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60, 61,62,63,64,65,66,67,68,69,70
lipa80b	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22, 23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41, 42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60, 61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80
lipa90b	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23, 24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43, 44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63, 64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83, 84,85,86,87,88,89,90

solution for all instances up to dimension $n = 40$. For the instances having dimension $n \geq 50$, the GRASP found, in at most 100,000 iterations, solutions within 1% of the known optimal value. On all instances, up to dimension $n = 90$, in problem class LIPAB, the GRASP found optimal solutions, taking a little over 13 CPU hours on the largest instance.

For all problem classes, the local search time accounted for less than half of the total CPU time, suggesting that a more ellaborate local search, such as the 3-exchange described in Subsection 2.3, can be used in the GRASP. In addition, GRASP may obtain better solutions if given more GRASP iterations. In this study, we limited the number of iterations to 100,000. For the few instances where GRASP did not produce a best known solution, a solution within a small percentage of the best known solution was found quickly.

4. Concluding remarks

In this paper, we discussed aspects of a GRASP implementation for solving the QAP. The algorithm was tested on a broad range of test problems and produced good-quality solutions in a reasonable amount of CPU time. Best known solutions were produced for almost all of the instances tested. In a few cases, the permutations found were better than those previously reported in the literature.

The algorithm can be easily implemented on a parallel computer since different GRASP iterations can be assigned to each processor. A single global variable (value of best solution found) is shared by the processors. It is expected that a parallel implementation should improve the computational results given in this paper as was observed in [**14**].

GRASP can be also adapted to solve special cases of QAP, like the traveling salesman problem, graph isomorphism, graph partitioning, and the band-width reduction problem.

References

[1] J.F. Bard and T.A. Feo, *Operations sequencing in discrete parts manufacturing*, Management Science **35** (1989), 249–255.

[2] J.F. Bard and T.A. Feo, *An algorithm for the manufacturing equipment selection problem*, IIE Transactions **23** (1991), 83–92.

[3] M.S. Bazaraa and H.D. Sherali, *Bender's partitioning scheme applied to a new formulation of the quadratic assignment problem*, Naval Research Logistics Quarterly **27** (1980), 29–41.

[4] S.H. Bokhari, *On the mapping problem*, IEEE Transactions on Computers **C-30** (1981), no. 3, 207–214.

[5] R.E. Burkard, S. Karisch, and F. Rendl, *QAPLIB – a quadratic assignment problem library*, European Journal of Operations Research **55** (1991), 115–119, Updated version – Feb. 1993.

[6] R.E. Burkhard and J. Offerman, *Entwurf von schreibmaschinentastaturen mittels quadratischer zuordnungsprobleme*, Z. Operations Res. **21** (1977), B121–B132.

[7] Paolo Carraresi and Federico Malucelli, *A new lower bound for the quadratic assignment problem*, Operations Research **40** (1992), no. Supplement 1, S22–S27.

[8] N. Christofides and E. Benavent, *An exact algorithm for the quadrtic assignment problem*, Operations Research **37** (1989), no. 5, 760–768.

[9] J.W. Dicky and J.W. Hopkins, *Campus building arrangement using TOPAZ*, Transportation Research **6** (1972), 59–68.

[10] A.W. Elshafei, *Hospital layout as a quadratic assignment problem*, Operations Research Quarterly **28** (1977), 167–179.

[11] B. Eschermann and H.J. Wunderlich, *Optimized synthesis of self-testable finite state machines*, 20th International Symposium on Fault-Tolerant Computing (FFTCS 20), 1990.

[12] T.A. Feo and J.F. Bard, *Flight scheduling and maintenance base planning*, Management Science **35** (1989), 1415–1432.

[13] T.A. Feo and M.G.C. Resende, *A probabilistic heuristic for a computationally difficult set covering problem*, Operations Research Letters **8** (1989), 67–71.

[14] T.A. Feo, M.G.C. Resende, and S.H. Smith, *A greedy randomized adaptive search procedure for the maximum independent set*, Tech. report, AT&T Bell Laboratories, Murray Hill, NJ, 1989, To appear in *Operations Research*.

[15] R.L. Francis and J.A. White, *Facility layout and location*, Prentice-Hall, Englewood Cliffs, N.J., 1974.

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

[17] L.J. Hubert, *Assignment methods in combinatorial data analysis*, Marcel Dekker, Inc., New York, NY 10016, 1987.

[18] D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis, *How easy is local search?*, Journal of Computer and System Sciences **37** (1988), 79–100.

[19] J.G. Klincewicz, *Avoiding local optima in the p-hub location problem using tabu search and* GRASP, Annals of Operations **40** (1992), 283–302.

[20] J.G. Klincewicz and A. Rajan, *Using GRASP to solve the component grouping problem*, Tech. report, AT&T Bell Laboratories, Holmdel, NJ, 1992.

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

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

[23] M. Laguna and J.L. González-Velarde, *A search heuristic for just-in-time scheduling in parallel machines*, Journal of Intelligent Manufacturing **2** (1991), 253–260.

[24] E.L. Lawler, *The quadratic assignment problem*, Management Science **9** (1963), 586–599.

[25] Y. Li, *Heuristic and exact algorithms for the quadratic assignment problem*, Ph.D. thesis, The Pennsylvania State University, 1992.

[26] Yong Li and Panos M. Pardalos, *Generating quadratic assignment test problems with known optimal permutations*, Computational Optimization and Applications **1** (1992), no. 2, 163–184.

[27] ———, Parallel algorithms for the quadratic assignment problem, in Recent Advances in Optimization and Parallel Computing, 177–189, Elsevier, Amsterdam, 1992, pp. 177–189.

[28] E.J. McCormick, *Human factors engineering*, McGraw-Hill, New York, 1970.

[29] K.A. Murthy, P.M. Pardalos, and Y. Li, *A local search algorithm for the quadratic assignment problem*, Informatica **3** (1992), no. 4, 524–538.

[30] C.E. Nugent, T.E. Vollmann, and J. Ruml, *An experimental comparison of techniques for the assignment of facilities to locations*, Journal of Operations Research **16** (1969), 150–173.

[31] P.M. Pardalos and J. Crouse, *A parallel algorithm for the quadratic assignment problem*, Proceedings of the Supercomputing 1989 Conference, ACM Press, 1989, pp. 351–360.

[32] C. Roucairol, *Du sequentiel au parallele: la recherche aborescente a la programmation quadratique en variables 0 et 1*, 1987, Thèse d'Etat.

[33] ———, *A parallel branch and bound algorithm for the quadratic assignment problem*, Discrete Applied Mathematics **18** (1987), 211–225.

[34] S. Sahni and T. Gonzalez, *P-complete approximation problems*, Journal of the Association of Computing Machinery **23** (1976), 555–565.

[35] A.A. Sch*ä*ffer and M. Yannakakis, *Simple local search problems that are hard to solve*, Tech. report, AT&T Bell Laboratories, 1989.

[36] L. Schrage, *A more portable fortran random number generator*, ACM Transactions on Mathematical Software **5** (1979), 132–138.

[37] M. Scriabin and R.C. Vergin, *Comparison of computers algorithms and visual based methods for plant layout*, Management Science **22** (1975), 172–187.

[38] H.D. Sherali and P. Rajgopal, *A flexible polynomial time construction and improvement heuristic for the quadratic assignment problem*, Computers & Operations Research **13** (1986), no. 5, 587–600.

[39] J. Skorin-Kapov, *Tabu search applied to the quadratic assignment problem*, ORSA Journal on Computing **2** (1990), no. 1, 33–45.

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

[41] M.R. Wilhelm and T.L. Ward, *Solving quadratic assignment problems by simulated annealing*, IEEE Transactions **19** (1987), no. 1, 107–119.

(Y. Li) Department of Computer Science, The Pennsylvania State University, University Park, PA 16802 USA

E-mail address, Y. Li: `yong@cs.psu.edu`

(P.M. Pardalos) Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611 USA and Technical University of Crete, Greece

E-mail address, P.M. Pardalos: `pardalos@math.ufl.edu`

(M.G.C. Resende) Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ 07974 USA

E-mail address, M.G.C. Resende: `mgcr@research.att.com`