INFORMS Journal on Computing
Vol. 12, No. 3, Summer 2000

0899-1499/100/1203-0164 $05.00

Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment

MARCELO PRAIS AND CELSO C. RIBEIRO / *Department of Computer Science, Catholic University of Rio de Janeiro, R. Marquês de São Vicente 225, Rio de Janeiro, RJ 22453-900, Brazil, Email: {prais, celso}@inf.puc-rio.br*

(Received: January 1998; revised: October 1998, May 1999; accepted: June 1999)

A greedy randomized adaptive search procedure (GRASP) is a metaheuristic for combinatorial optimization. In this paper, we describe a GRASP for a matrix decomposition problem arising in the context of traffic assignment in communication satellites. We review basic concepts of GRASP: construction and local search algorithms. The local search phase is based on the use of a new type of neighborhood defined by constructive and destructive moves. The implementation of a GRASP for the matrix decomposition problem is described in detail. Extensive computational experiments on literature and randomly generated problems are reported. Moreover, we propose a new procedure *Reactive GRASP*, in which the basic parameter that defines the restrictiveness of the candidate list during the construction phase is self-adjusted according to the quality of the solutions previously found. The approach is robust and does not require calibration efforts. On most of the literature problems considered, the new *Reactive GRASP* heuristic matches the optimal solution found by an exact column-generation with branch-and-bound algorithm.

We consider in this work a matrix decomposition problem arising in the context of traffic scheduling in satellite-switched time-division-multiple-access (SS/TDMA) systems. A geostationary communication satellite operating under SS/TDMA mode has a number of spot beam antennas that cover geographically distributed zones. According to the slot-switching configuration of the onboard switch, the uplink traffic received at the satellite has to be immediately sent to ground zones through a set of transponders. The number of available transponders determines the maximum number of connections that can be simultaneously established through the satellite. The slot-switching configurations are determined through the solution of a time slot assignment problem, which is equivalent to the decomposition of a nonnegative traffic matrix into the sum of a family of switching-mode matrices. Several variants of this problem have been extensively studied in the literature, taking into account different objective functions and constraint configurations.[1–5, 7, 9–11, 12, 15]

Let $T = \{t_{ij}\}_{i=1,\ldots,n}^{j=1,\ldots,n}$ be a $n \times n$ traffic matrix in which each entry t_{ij} represents the amount of traffic to be sent (or the connection duration) from a transmitting antenna i to a receiving antenna j. We consider the problem of determining an optimal sequence of slot switching configurations, minimizing the overall time needed to transmit the whole traffic matrix T through a SS/TDMA satellite using no more than n transponders without allowing preemption (i.e., whenever two antennas i and j are connected, they cannot be disconnected until after all traffic t_{ij} from i to j is sent). This problem has been shown to be NP-hard by Rendl[17] (see also Refs. 8, 9, and 16 for other complexity results).

In this paper we present a greedy randomized adaptive search procedure (GRASP) for the above problem. Moreover, we extend the basic ideas of GRASP to propose a new procedure *Reactive GRASP*, whose basic parameter is self-adjusted according to the quality of the solutions found. In Section 1, we give the detailed formulation of the matrix decomposition (or time slot assignment) problem, together with a simple, purely greedy heuristic for its approximate solution. Section 2 describes a greedy randomized adaptive search procedure for the time slot assignment problem. In Section 3, we present computational results on randomly generated and literature problems, illustrating the efficiency of the GRASP approach. In Section 4, we propose a new procedure called *Reactive GRASP*, in which the restricted candidate list parameter is self-adjusted according to the quality of the solutions found, and we show that it further improves the computational results. Concluding remarks are made in Section 5.

1. Formulation of the Time Slot Assignment Problem

The time slot assignment problem described in the previous section may be viewed as a special case of a scheduling problem under purely disjunctive constraints. Given the traffic matrix $T = \{t_{ij}\}_{i=1,\ldots,n}^{j=1,\ldots,n}$ with m strictly positive entries, it consists in finding a decomposition of T into a sum of q (switching mode) matrices, i.e.,

$$T = \sum_{k=1}^{q} P^k,$$

meeting the following conditions:

- for all $k = 1, \ldots, q$, $P^k = \{p_{ij}^k\}_{i=1,\ldots,n}^{j=1,\ldots,n}$ is a switching mode matrix, i.e., it has no more than one nonzero entry in each row or column;
- for all $k = 1, \ldots, q$, $p_{ij}^k > 0 \Rightarrow p_{ij}^k = t_{ij}$, $\forall i = 1, \ldots, n$, $j = 1, \ldots, n$, i.e., there is no preemption and each nonzero

Subject classifications: Combinatorial optimization.
Other key words: matrix decomposition, traffic assignment, TDMA, local search, GRASP, reactive GRASP.

```
procedure ConstructGreedySolution(T)
1    k ← 1; P^1 ← 0;
2    while there are unselected nonzero entries in the original traffic matrix do
3        if there is at least one yet unselected nonzero entry (i, j) of matrix T
         such that X(P^k, i, j) = 1
4        then do
5            Let t_{i*j*} be the largest among all such entries;
6            p^k_{i*j*} ← t_{i*j*};
7        end then;
8        else do
9            k ← k + 1; P^k ← 0;
10       end else;
11   end while;
12   q ← k;
13   return S = {P^k, k = 1, ..., q};
end ConstructGreedySolution;
```

Figure 1. Pseudo-code of the purely greedy construction procedure.

entry of the original traffic matrix should appear in one and exactly one of the matrices of the decomposition; and

- the objective function $F(P^1, \ldots, P^q) = \sum_{k=1}^{k=q} c_{max}(P^k)$ is minimized, where $c_{max}(P^k) = \max_{i=1,\ldots,n;j=1,\ldots,n}\{p_{ij}^k\}$ denotes the L_∞ norm of matrix P^k, $k = 1, \ldots, q$.

We notice that the total number q of switching mode matrices appearing in the decomposition is not imposed. However, it is also possible to refer to the case where it is fixed: just take q equal to any upper bound on the number of matrices appearing in the decomposition (e.g., $q = m$, the number of nonzero entries in T) and allow some of the matrices P^k to be null. This matrix decomposition problem may be formulated as a very large set partitioning problem (e.g., see Refs. 1, 14, and 19). Ribeiro et al.[19] used this formulation to exactly solve the problem using a branch-and-bound procedure combined with column generation and a ranking technique. Computational results are reported for literature problems involving traffic matrices with dimensions up to $n = 15$.

Given an $n \times n$ switching mode matrix P, a row index $i \in \{1, \ldots, n\}$, and a column index $j \in \{1, \ldots, n\}$, let $X(P, i, j)$ be the following compatibility function:

$$X(P, i, j) = \begin{cases} 1, & \text{if all elements in row } i \text{ and all elements in column } j \text{ of matrix } P \text{ are zero,} \\ 0, & \text{otherwise.} \end{cases}$$

Then, if $X(P, i, j) = 1$, a nonzero entry can be inserted at position (i, j) of mode matrix P without violating the one-element-per-row-and-column restriction. A very simple greedy heuristic for the time slot assignment problem is described in the pseudo-code of Fig. 1. The procedure takes as input the original traffic matrix T to be decomposed. The decomposition is initialized in line 1. The loop from line 2 to line 11 assigns each positive entry of the original traffic matrix T to exactly one of the switching mode matrices P^k, $k = 1, \ldots, q$ in the decomposition. In line 3, we make use of the compatibility function above defined to examine if there are unassigned positive entries of T compatible with the current switching mode matrix P^k being constructed (i.e., which do not violate the one-element-per-row-and-column restriction). If such an entry exists, in line 5 we determine the largest one among all possible candidates, and in line 6 we insert it into P^k. Otherwise, in case all unassigned entries conflict with P^k, a new switching mode matrix is initialized in line 9 to accommodate them. The number q of matrices into which T was decomposed is set at line 12, and the solution $S = \{P^k, k = 1, \ldots, q\}$ formed by the matrices appearing in the decomposition is returned in line 13.

We now evaluate the complexity of procedure ConstructGreedySolution. First, the m positive entries of T are sorted in non-decreasing order in $O(m \log m) = O(n^2 \log n)$ time. This is followed by an $O(m)$ scan of the sorted list for each mode matrix in the decomposition, leading to an additional $O(qm) = O(qn^2)$ time. Then, the overall complexity amounts to $O(n^2(q + \log n))$, which is roughly $O(qn^2)$ because, in practice, $q \approx n$.

Example: Below we give an example of the application of the greedy heuristic. Let the traffic matrix be

$$T = \begin{bmatrix} 30 & 40 & 15 \\ 0 & 0 & 60 \\ 20 & 15 & 10 \end{bmatrix}.$$

We start the construction of the first mode matrix by placing the largest nonzero entry $t_{23} = 60$ into P^1. The next largest entry $t_{12} = 40$ is also compatible with P^1. Since $t_{11} = 30$ is not compatible with P^1, we complete its construction by inserting $t_{31} = 20$, which is the largest remaining nonzero entry. We repeat the same steps, placing the largest yet unselected entry $t_{11} = 30$ in P^2, and so on. At the end, we obtain the decomposition $T = P^1 + P^2 + P^3 + P^4$, with:

$$P^1 = \begin{bmatrix} 0 & 40 & 0 \\ 0 & 0 & 60 \\ 20 & 0 & 0 \end{bmatrix}, P^2 = \begin{bmatrix} 30 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 15 & 0 \end{bmatrix},$$

$$P^3 = \begin{bmatrix} 0 & 0 & 15 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \text{ and } P^4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 10 \end{bmatrix}.$$

The cost of this solution is $F(P^1, P^2, P^3, P^4) = c_{max}(P^1) + c_{max}(P^2) + c_{max}(P^3) + c_{max}(P^4) = 60 + 30 + 15 + 10 = 115$.

2. A GRASP for Time Slot Assignment

In this section, we apply the concepts of GRASP to the time slot assignment (or matrix decomposition) problem. A GRASP[6] is an iterative process in which each GRASP iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is explored by local search. The best solution over all GRASP iterations is returned as the result.

We summarize below the basic concepts of GRASP, as presented in Resende and Ribeiro.[18] In the construction phase, a feasible solution is built, one element at a time. At each construction iteration, the next element to be added is determined by ordering all elements in a candidate list with

```
procedure GRASP(ListSize,max_iterations,RandomSeed)
1   InputInstance();
2   for k = 1,..., max_iterations do
3       ConstructGreedyRandomizedSolution(ListSize,RandomSeed);
4       LocalSearch(BestSolutionFound);
5       UpdateSolution(BestSolutionFound);
6   end for;
7   return BestSolutionFound;
end GRASP;
```

Figure 2. A generic GRASP pseudo-code.

respect to a greedy function that estimates the benefit of selecting each 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 one.

The solutions generated by a GRASP construction are not guaranteed to be locally optimal. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better one from its neighborhood. It terminates when there are no better solutions in the neighborhood. Success for a local search algorithm depends on 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.

GRASP can be seen as a metaheuristic that captures good features of pure greedy algorithms (e.g., fast local search convergence and good quality solutions) and also of random construction procedures (e.g., diversification). Fig. 2 illustrates a generic GRASP implementation in pseudo-code. The algorithm takes as input parameters the candidate list size, the maximum number of GRASP iterations, and the seed for the random number generator. We describe in the next subsection how the greedy heuristic procedure ConstructGreedySolution presented in Section 1 may be used to yield a randomized greedy algorithm that will constitute the construction phase of a GRASP to the time slot assignment problem.

2.1 Construction Phase

Algorithm ConstructGreedyRandomizedSolution outlined in the pseudo-code of Fig. 3 takes as input the original traffic matrix T to be decomposed, the restricted candidate list (RCL) parameter α ($0 \leq \alpha \leq 1$), and a seed for the pseudo random number generator. The decomposition is initialized in line 1. The loop from line 2 to line 12 assigns each positive entry of the original traffic matrix T to exactly one of the switching mode matrices P^k, $k = 1, \ldots, q$, in the decomposition. In line 3 we examine if there are unassigned positive entries of T compatible with the current switching mode matrix P^k being constructed (i.e., which do not violate the one-element-per-row-and-column restriction). If such an entry exists, in line 5 we determine the maximum value t_{max} among all entries satisfying this condition. Next, in line 6 all yet unselected entries (i, j) compatible with the current switching mode matrix P^k being constructed whose values t_{ij}

```
procedure ConstructGreedyRandomizedSolution(T,α,seed)
1   k ← 1; P^1 ← 0;
2   while there are unselected nonzero entries in the original traffic matrix T do
3       if there is at least one yet unselected nonzero entry (i,j) of matrix T
        such that X(P^k,i,j) = 1
4       then do
5           t_max ← max{t_ij, i,j = 1,...,n : t_ij > 0 is unselected and
                        X(P^k,i,j) = 1};
6           RCL ← {(i,j) : t_ij ∈ [t_max(1 − α), t_max] is unselected and
                        X(P^k,i,j) = 1};
7           (i*,j*) ← random(seed, RCL);
8           p^k_{i*j*} ← t_{i*j*};
9       end then;
10      else do
11          k ← k + 1; P^k ← 0;
12      end else;
13  end while;
14  q ← k;
15  return S = {P^k, k = 1,...,q};
end ConstructGreedyRandomizedSolution;
```

Figure 3. Pseudo-code of GRASP construction phase.

are in the range $[(1 - \alpha)t_{max}, t_{max}]$ are placed in the RCL. A single entry is selected at random from the list in line 7, and in line 8 we insert it into P^k. Otherwise, in case all unassigned entries conflict with P^k, a new switching mode matrix is initialized in line 11 to accommodate them. The number q of matrices into which T was decomposed is set at line 14 and the solution $S = \{P^k, k = 1, \ldots, q\}$ formed by the matrices appearing in the decomposition is returned in line 15.

2.2 Local Search

Since the solution produced by the construction phase is not necessarily a local optimum, local search can be applied as an attempt to improve it. The first step towards the implementation of a local search procedure consists in identifying an appropriate neighborhood definition. Commonly used neighborhoods, such as pairwise exchanges, are not suitable for the time slot assignment problem since they usually do not lead to feasible neighbors. To illustrate it, let $p^{k_1}_{ab}$ and $p^{k_2}_{cd}$ be two entries to be exchanged, respectively, from matrices P^{k_1} and P^{k_2}. Because in many cases matrices P^{k_1} and P^{k_2} will have nonzero entries in most of their rows and columns, neither element t_{ab} will be compatible with P^{k_2} or element t_{cd} with matrix P^{k_1}. Although Dell'Amico, Maffioli, and Trubian[4] have used this kind of neighborhood for a similar problem, they pointed out themselves[13] that, for their version of the time slot assignment problem, many rows and columns do not have any nonzero entry, making it easy to generate a new feasible decomposition through the exchange of pairs of nonzero entries from two matrices appearing in the decomposition.

We propose a new type of neighborhood for this problem. Instead of defining the neighborhood itself, we describe in Fig. 4 the pseudo-code of the local search algorithm used to generate a set of moves that lead to neighbors of the current solution $S = \{P^k, k = 1, \ldots, q\}$. The best neighbor S' of the current solution S is returned. Procedure LocalSearch basically makes use of two types of moves. Insertion moves are based on taking each mode matrix P^k in the current decomposition with strictly less than n elements in turn, filling up

```
procedure LocalSearch(T,S = {P^k, k = 1,...,q},max_permutations,
        max_eliminations,seed)
1   S' ← S; best_neighbor_value ← F(S);
2   for k = 1,...,q do
3       if P^k has strictly less than n nonzero entries
4       then S^k ← InsertMoves(T,P^k,max_permutations,seed);
5       else S^k ← DeleteMoves(T,P^k,max_eliminations,seed);
6       if F(S^k) < best_neighbor_value
7       then do
8           S' ← S^k;
9           best_neighbor_value ← F(S^k);
10      end then;
11  end for;
12  return S';
end LocalSearch;
```

Figure 4. Pseudo-code of local search phase.

P^k as much as possible by adding nonzero entries of the original traffic matrix to it, and recomputing a decomposition by applying the greedy procedure ConstructGreedySolution described in Fig. 1 to the remaining nonzero entries of the original traffic matrix. Analogously, elimination moves are based on taking each complete mode matrix in the current decomposition with exactly n nonzero entries in turn, eliminating some of them, and recomputing a decomposition as above.

Insertion moves are generated and evaluated by procedure InsertMoves, which takes as input the original traffic matrix T, the mode matrix P^k under investigation for insertion moves, a parameter max_permutations which defines the maximum number of neighbors that will be evaluated, and a seed for the pseudo random number generator. In line 1 we create a list L with all nonzero entries of the traffic matrix T that could be inserted into P^k without violating the one-element-per-row-and-column restriction with respect to P^k. In line 2 we initialize the value of the best neighbor associated with insertion moves from mode matrix P^k. Within the loop from line 3 to line 21, up to max_permutations neighbors of the current decomposition are generated and evaluated. To do so, in line 4 we initialize the new mode matrix $\bar{P}$, which will replace P^k in the neighbor being constructed. Each neighbor is associated with a random permutation L' of list L that is computed in line 5. The loop from line 6 to line 10 searches list L' for nonzero entries compatible with the new mode matrix $\bar{P}$ under construction. Line 6 is used to initialize the loop. The l-th element (i, j) of list L is identified in line 7 and checked for feasibility in line 8. If it does not violate the one-element-per-row-and-column restriction with respect to $\bar{P}$, then it is inserted into the latter in line 9. At the end of this loop, a new mode matrix $\bar{P}$ has been constructed and will be used as the basis for the computation of a neighbor solution. In line 11, we compute the partial traffic matrix $\bar{T}$ obtained from T by the elimination of the nonzero entries already assigned to $\bar{P}$. The greedy algorithm ConstructGreedySolution is applied in line 12 to find a decomposition $\bar{S}$ of this partial traffic matrix $\bar{T}$ into $\bar{q}$ mode matrices $\bar{P}^s$, $s = 1, \ldots, \bar{q}$. The neighbor under generation is completed in lines 13–15 by appending matrix $\bar{P}$ as the $(\bar{q} + 1)$-th one in the decomposition. In line 16, we compare the cost of the neighbor $\bar{S}$ just generated with the value of the best neighbor obtained by insertion moves so far. In case the former is better, we update the best neighbor and the best insertion neighbor value in lines 18 and 19. The best neighbor S^k found is returned in line 22.

Example: We illustrate below an example of the generation and evaluation of insertion moves, taking the same traffic matrix T and the decomposition $S = \{P^1, P^2, P^3, P^4\}$ obtained with the greedy algorithm, as described in Section 1. The cost of this solution is $F(S) = 60 + 30 + 15 + 10 = 115$. We consider the neighbors being generated by insertions into

$$P^4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 10 \end{bmatrix}.$$

The list of elements of the original traffic matrix T compatible with P^4 is $L = \{(1, 1), (1, 2)\}$. Only two permutations of L can be constructed: $(1, 1) - (1, 2)$ and $(1, 2) - (1, 1)$. If we consider the first permutation, only element (1, 1) may be inserted in P^4 and we obtain

$$\bar{P} = \begin{bmatrix} 30 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 10 \end{bmatrix}.$$

The application of the greedy algorithm to $\bar{T} = T - \bar{P}$ leads to the completion of the decomposition with mode matrices

$$\begin{bmatrix} 0 & 40 & 0 \\ 0 & 0 & 60 \\ 20 & 0 & 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & 0 & 15 \\ 0 & 0 & 0 \\ 0 & 15 & 0 \end{bmatrix},$$

corresponding to an improving neighbor with cost 105. If we consider the second permutation, element (1, 2) would be inserted into P^4, leading to a neighbor formed by mode matrices

$$\begin{bmatrix} 30 & 0 & 0 \\ 0 & 0 & 60 \\ 0 & 15 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 15 \\ 0 & 0 & 0 \\ 20 & 0 & 0 \end{bmatrix}, \text{ and } \begin{bmatrix} 0 & 40 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 10 \end{bmatrix},$$

whose cost is 120, larger than that of the current solution. Then, for $k = 4$, procedure InsertMoves returns the best neighbor

$$S^4 = \begin{bmatrix} 0 & 40 & 0 \\ 0 & 0 & 60 \\ 20 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 15 \\ 0 & 0 & 0 \\ 0 & 15 & 0 \end{bmatrix}, \text{ and } \begin{bmatrix} 30 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 10 \end{bmatrix},$$

with cost $F(S^4) = 105$.

Elimination moves are generated and evaluated by procedure DeleteMoves, which takes as input the original traffic

matrix T, the mode matrix P^k under investigation for elimination moves, a parameter max_eliminations which defines the maximum number of neighbors that will be evaluated, and a seed for the pseudo random number generator. In line 1, we create a list L with the n nonzero entries of P^k. In line 2, we initialize the value of the best neighbor associated with elimination moves from mode matrix P^k. Within the loop from line 3 to line 18, max_eliminations neighbors of the current decomposition are generated and evaluated. To do so, in line 4 we initialize the new mode matrix $\bar{P}$, which will replace P^k in the neighbor being constructed. We choose an element of L at random in line 5 to be eliminated from the current mode matrix P^k in line 6. The new mode matrix $\bar{P}$ so constructed will be used as the basis for the computation of a neighbor solution. List L is updated in line 7. In line 8, we compute the partial traffic matrix $\bar{T}$ obtained from T by the elimination of the nonzero entries already assigned to $\bar{P}$. The greedy algorithm ConstructGreedySolution is applied in line 9 to find a decomposition $\bar{S}$ of this partial traffic matrix $\bar{T}$ into $\bar{q}$ mode matrices $\bar{P}^s$, $s = 1, \ldots, \bar{q}$. The neighbor under generation is completed in lines 10–12 by appending matrix $\bar{P}$ as the $(\bar{q} + 1)$-th one in the decomposition. In line 13, we compare the cost of the neighbor $\bar{S}$ just generated with the value of the best neighbor obtained by elimination moves so far. In case the former is better, we update the best neighbor and the best elimination neighbor value in lines 15 and 16. The best neighbor S^k found is returned in line 19.

Example: Below we present an example of the generation and evaluation of elimination moves, taking again the same traffic matrix T and the decomposition $S = \{P^1, P^2, P^3, P^4\}$ obtained with the greedy algorithm, as described in Section 1. We consider the neighbors being generated by eliminations from

$$P^1 = \begin{bmatrix} 0 & 40 & 0 \\ 0 & 0 & 60 \\ 20 & 0 & 0 \end{bmatrix}.$$

Let us take max_eliminations $= 1$ and eliminate element $p^1_{31} = 20$, leading to the new mode matrix

$$\bar{P} = \begin{bmatrix} 0 & 40 & 0 \\ 0 & 0 & 60 \\ 0 & 0 & 0 \end{bmatrix}.$$

The application of the greedy algorithm to $\bar{T} = T - \bar{P}$ leads to the completion of the decomposition with mode matrices

$$\begin{bmatrix} 30 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 15 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 15 \\ 0 & 0 & 0 \\ 20 & 0 & 0 \end{bmatrix}, \text{ and } \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 10 \end{bmatrix},$$

corresponding to a worst neighbor with cost 120.

According to the above description and to the pseudo-codes in Figs. 4, 5, and 6, procedure ConstructGreedySolution described in Section 1 is applied many times during the neighborhood generation and evaluation. Since it has complexity $O(qn^2)$, the overall complexity of finding the best neighbor of the current solution is $O(tq^2n^2)$, where q is the number of mode matrices in the current solution and $t = \max\{\text{max\_permutations}, \text{max\_eliminations}\}$ is an upper bound to the number of neighbors generated from each of them. Since in practice $q \approx n$, this amounts to $O(tn^4)$. Due to this high computational cost, we implemented procedure LocalSearch as outlined in Fig. 4, just as the search for the best neighbor S' of the current solution and not as a complete local search seeking a local optimum, which could be attained from S. The correctness of this choice was confirmed by the computational experiments reported later in

```
procedure InsertMoves(T, P^k,max_permutations,seed)
1   L ← {(i,j), i,j = 1,...,n : t_ij > 0, p^k_ij = 0 and X(P^k,i,j) = 1};
2   best_insertion_value ← ∞;
3   for r = 1,...,max_permutations do
4       P̄ ← P^k;
5       L' ← RandomPermutation(L);
6       for ℓ = 1,...,| L' | do
7           Let (i,j) be the ℓ-th element of L';
8           if X(P̄,i,j) = 1
9           then p̄_ij ← t_ij;
10      end for;
11      T̄ ← T − P̄;
12      S̄ = {P̄^s, s = 1,...,q̄} ← ConstructGreedySolution(T̄);
13      P̄^(q̄+1) ← P̄;
14      S̄ ← S̄ ∪ P̄^(q̄+1);
15      q̄ ← q̄ + 1;
16      if F(S̄) < best_insertion_value
17      then do
18          S^k ← S̄;
19          best_insertion_value ← F(S^k);
20      end then;
21  end for;
22  return S^k;
end InsertMoves;
```

Figure 5. Pseudo-code of the generation and evaluation of insertion moves.

```
procedure DeleteMoves(T, P^k,max_eliminations,seed)
1   L ← {(i,j), i,j = 1,...,n : p^k_ij > 0};
2   best_elimination_value ← ∞;
3   for r = 1,...,max_eliminations while L ≠ ∅ do
4       P̄ ← P^k;
5       (i,j) ← random(seed,L);
6       p̄_ij ← 0;
7       L ← L − {(i,j)};
8       T̄ ← T − P̄;
9       S̄ = {P̄^s, s = 1,...,q̄} ← ConstructGreedySolution(T̄);
10      P̄^(q̄+1) ← P̄;
11      S̄ ← S̄ ∪ P̄^(q̄+1);
12      q̄ ← q̄ + 1;
13      if F(S̄) < best_elimination_value
14      then do
15          S^k ← S̄;
16          best_elimination_value ← F(S^k);
17      end then;
18  end for;
19  return S^k;
end DeleteMoves;
```

Figure 6. Pseudo-code of the generation and evaluation of elimination moves.

Table I. Run Statistics on Random Instances with $\alpha = 0.1$, 0.3, 0.5, and 0.0

		$\alpha = 0.1$			$\alpha = 0.3$			$\alpha = 0.5$			$\alpha = 0.0$
Problem	n	Best	ITR	Secs	Best	ITR	Secs	Best	ITR	Secs	Best
R.01	15	6724	8	377	6724	41	377	6724	601	376	6760
R.02	15	6771	1	349	6771	100	353	6771	279	362	6771
R.03	15	6475	693	351	6504	707	352	6499	306	354	6518
R.04	15	6342	7	385	6344	410	378	6361	932	366	6388
R.05	15	6415	626	349	6435	313	352	6440	582	349	6477
R.06	15	7550	1	412	7550	1	420	7550	1	423	7550
R.07	15	6889	55	430	6900	97	416	6903	258	410	6900
R.08	15	6076	586	334	6076	939	341	6103	60	340	6145
R.09	15	6434	41	336	6422	7	336	6428	977	336	6510
R.10	15	6525	5	382	6525	410	384	6527	264	384	6540
Average	15	6620	202	371	6625	303	371	6631	426	370	6656
R.11	18	7682	135	914	7713	265	900	7706	902	891	7736
R.12	18	7693	580	855	7702	880	837	7707	112	796	7698
R.13	18	7524	718	957	7545	198	928	7556	691	929	7546
R.14	18	8254	65	841	8254	478	856	8259	425	871	8270
R.15	18	8021	1	951	8021	1	937	8021	1	957	8021
R.16	18	7755	30	917	7755	253	909	7755	982	920	7764
R.17	18	8032	222	1017	8025	828	974	8025	273	949	8033
R.18	18	7234	156	792	7264	877	792	7297	684	805	7322
R.19	18	6858	947	720	6876	409	693	6922	76	702	7086
R.20	18	8027	183	1054	8028	259	1034	8034	850	1049	8055
Average	18	7708	304	902	7718	445	886	7728	500	887	7753
R.21	21	9287	769	1821	9290	4	1813	9290	48	1829	9290
R.22	21	8890	958	1972	8893	585	1955	8896	249	1949	8914
R.23	21	9145	906	2095	9142	93	2036	9145	169	2084	9187
R.24	21	8464	491	1784	8473	24	1692	8476	239	1724	8464
R.25	21	8511	855	1742	8508	135	1703	8520	215	1730	8535
R.26	21	8838	172	2129	8852	216	2065	8880	690	2050	9000
R.27	21	7980	108	1729	7980	597	1638	7994	530	1646	7980
R.28	21	9407	4	2393	9407	3	2305	9407	14	2380	9407
R.29	21	8693	514	1805	8696	957	1804	8706	286	1819	8713
R.30	21	9816	45	2398	9823	589	2366	9817	985	2385	9816
Average	21	8903	482	1988	8906	320	1938	8913	343	1960	8931

Table II, in which different strategies for the implementation of the local search are compared. The pseudo-code with the complete description of procedure GRASP_for_TSA for the time slot assignment problem is given in Fig. 7.

3. Experimental Results

In this section, we present experimental results obtained with a Fortran implementation of the GRASP for time slot assignment GRASP_for_TSA shown in Fig. 7. Randomly generated test problems were used in the first phase of the computational experiments to set the best strategy for our GRASP. In the second phase, such best strategy is applied to a set of realistic literature problems whose optimal solutions are known. The computational experiments have been conducted on an IBM 9672 model R34 mainframe computer. The code was written in Fortran 77 and compiled with the IBM VS Fortran compiler (version 2, release 5).

3.1 Random Problems

A set of 30 randomly generated test problems was used in the first phase of the experiments. For each value of $n = 15$, 18, and 21, we generated 10 test problems. The density of the traffic matrices was taken as 63% (which is also the average density of the literature problems reported and exactly solved by Ribeiro et al.[19]). The nonzero entries are integers and have been generated according to a uniform distribution between 1 and 800. Throughout the computational experiments with random instances reported in this section, we have always performed 1000 GRASP iterations (i.e., max_iterations = 1000).

We first ran GRASP_for_TSA with the restricted candidate list parameter $\alpha = 0.1$, 0.3, 0.5, and 0.0 (greedy choice), max_permutations equal to the number of elements in the list L, and max_eliminations = 1. Statistics of the computational results are displayed in Table II. For each test prob-

Table II. Run Statistics on Random Instances Comparing Local Search Approaches

		BN		II		SN	
Problem	n	Best	Secs	Best	Secs	Best	Secs
R.01	15	6724	377	6724	470	6724	887
R.02	15	6771	349	6771	575	6771	1041
R.03	15	6475	351	6476	606	6475	966
R.04	15	6342	385	6342	504	6342	989
R.05	15	6415	349	6409	502	6409	904
R.06	15	7550	412	7550	498	7550	899
R.07	15	6889	430	6889	644	6889	1145
R.08	15	6076	334	6075	413	6076	810
R.09	15	6434	336	6420	459	6424	868
R.10	15	6525	382	6525	566	6525	1048
Average	15	6620	371	6618	524	6618	956
R.11	18	7682	914	7679	1197	7669	2423
R.12	18	7693	855	7691	1086	7691	2267
R.13	18	7524	957	7530	1151	7524	2365
R.14	18	8254	841	8254	1160	8254	2342
R.15	18	8021	951	8021	1331	8021	2395
R.16	18	7755	917	7755	1387	7755	2696
R.17	18	8032	1017	8016	1529	8025	3175
R.18	18	7234	792	7234	1014	7232	1902
R.19	18	6858	720	6871	1158	6858	2082
R.20	18	8027	1054	8027	1799	8027	3452
Average	18	7708	902	7708	1281	7706	2510
R.21	21	9287	1821	9289	2420	9287	4948
R.22	21	8890	1972	8887	2481	8887	5603
R.23	21	9145	2095	9141	2712	9145	5887
R.24	21	8464	1784	8464	2398	8464	4997
R.25	21	8511	1742	8505	2421	8504	5139
R.26	21	8838	2129	8829	2581	8819	5462
R.27	21	7980	1729	7975	2556	7975	5064
R.28	21	9407	2393	9407	2580	9407	5874
R.29	21	8693	1805	8693	2501	8693	4923
R.30	21	9816	2398	9816	2567	9816	6145
Average	21	8903	1988	8901	2522	8900	5404

lem, this table lists the dimension of the traffic matrix (n), the value of the best solution found (best), the iteration on which the best solution was found (itr), and the computation time in seconds (secs) for the 1000 iterations. Average results for the 10 instances associated with each problem size are also presented.

We observe from the results in Table II that smaller values of the RCL parameter α seem to lead to better results. For all 30 randomly generated instances, the best solution was almost always found with $\alpha = 0.1$. Also, $\alpha = 0.3$ performed better than $\alpha = 0.5$. Based on these results, we set $\alpha = 0.1$ throughout the additional computational experiments reported below. We also notice that this setting leads to systematically better solutions than those obtained with the greedy choice followed by local search ($\alpha = 0.0$; last column of Table II).

To investigate the behavior of the local search procedure, we implemented three versions of it. The first version reproduces algorithm LocalSearch outlined in Fig. 4, in which we only investigate the neighborhood of the solution constructed by procedure ConstructGreedyRandomizedSolution, even if an improving move was found. The second version corresponds to an iterative improvement algorithm, which always moves to the first found improving neighbor of the current solution. The third implementation is a steepest-descent approach, which examines the whole neighborhood and moves to the best neighbor. These three versions of the local search procedure are referred to respectively as BN (for best neighbor of the constructed solution), II (for iterative improvement), and SD (for steepest descent). Computational results comparing the results obtained by these three approaches for the 30 randomly generated instances are presented in Table I. For each test problem, and for each of these three versions, this table lists the dimension of the

```
procedure GRASP_for_TSA(T,α,seed,max_iterations,
          max_permutations,max_eliminations)
1   best_value ← ∞;
2   for k = 1,...,max_iterations do
3       S ← ConstructGreedyRandomizedSolution(T,α,seed);
4       S' ← LocalSearch(T,S,max_permutations,max_eliminations,
            seed);
5       if F(S') < best_value
6       then do
7           S* ← S';
8           best_value ← F(S*);
9       end then;
10  end for;
11  return S*;
end GRASP_for_TSA;
```

Figure 7. Pseudo-code of GRASP for time slot assignment.

```
procedure GRASP_with_filter_for_TSA(T,α,seed,max_iterations,
          max_permutations,max_eliminations)
1   best_value ← ∞;
2   sum ← 0; times ← 0;
3   for k = 1,...,max_iterations do
4       S ← ConstructGreedyRandomizedSolution(T,α,seed);
5       if k ≥ 100                                        /* filter test */
6       then if 0.9 × (1 − β).F(S) > best_value
7            then go to line 17;
8       S' ← LocalSearch(T,S,max_permutations,max_eliminations,
            seed);
9       if F(S') < best_value                  /* best solution update */
10      then do
11          S* ← S';
12          best_value ← F(S*);
13      end then;
14      times ← times + 1;                      /* filter computations */
15      sum ← sum + (F(S) − F(S'))/F(S);
16      β ← sum/times;
17  end for;
18  return S*;
end GRASP_with_filter_for_TSA;
```

Figure 8. Pseudo-code of the GRASP with filter for time slot assignment.

traffic matrix (n), the value of the best solution found (best), and the computation time in seconds (secs) for the 1000 iterations. Average results for each problem size are also given. We can see that the computation times observed for the BN (best neighbor of the constructed solution) version of the local search are considerably smaller on the average than those obtained with II and SD, with a very small loss in terms of solution quality. For this reason, we decided to use the original LocalSearch procedure for the local search phase of our GRASP, as presented in Section 2.

The most time-consuming phase of algorithm GRASP_for_TSA is certainly the local search, since it is based on the successive application of procedure ConstructGreedySolution, which has the same computational complexity as procedure ConstructGreedyRandomizedSolution applied only once during the construction phase. To speed up algorithm GRASP_for_TSA, we have then investigated the use of a filter to eliminate runs of LocalSearch originating from unpromising bad solutions generated by the construction phase. To do so, we store the average value β of the ratio $(F(S) - F(S'))/F(S)$, i.e., the average reduction obtained by the local search phase with respect to the solution constructed in the first phase. After the first 100 iterations, we make use of this information to decide whether each constructed solution will be submitted to local search or not. The idea is based on the rationale that if some reasonable threshold applied to the cost of the constructed solution leads to a value much higher than the cost of the best solution already found, it is unlikely that LocalSearch could produce a better solution than the current best. We use 90% of $(1 - \beta)$ as this threshold, to give more chances to the local search. The pseudo-code of algorithm GRASP_with_filter_for_TSA for this extended version of the GRASP described in Section 2 is outlined in Fig. 8.

We ran both algorithms GRASP_for_TSA and GRASP_with_filter_for_TSA with the same parameter settings. Statistics of the computational results obtained for the 30 randomly generated instances are displayed in Table III. For each test problem, and for each of these algorithms, this table lists the dimension of the traffic matrix (n), the value of the best solution found (best), the index of the iteration on which the best solution was found (itr), the number of times LocalSearch is effectively performed (#_LS), and the computation time in seconds (secs). Average results for each problem size are also given.

In a final step to tune the best strategy for our GRASP, we evaluated a variant of GRASP_with_filter_for_TSA in which one considers only two permutations of the list L of possible insertion moves. Statistics of the computational results obtained for the 30 randomly generated instances are displayed in Table IV. For each test problem, and for each of these algorithms, this table lists the dimension of the traffic matrix (n), the value of the best solution found (best), the index of the iteration on which the best solution was found (itr), the number of times LocalSearch is effectively performed (#_LS), and the computation time in seconds (secs). Average results for each problem size are also given. We notice that GRASP_with_filter_for_TSA using max_permutations = 2 performs better than with the previous setting, since it achieves significant reductions in computation times while obtaining solutions with the same overall quality.

3.2 Literature Problems

Further computational experiments have been performed with the application of algorithm GRASP_with_filter_for_TSA with the final parameter settings (max_permutations = 2) to a set of literature problems exactly solved by Ribeiro et al.[19] We report computational results obtained with our GRASP approach for 36 of the 42 original instances (we discarded very small instances with $n < 5$ as well as problem P.39 whose data were not available). Computational results obtained with max_iterations = 1000 and all other parameter settings as described above are summarized in Table V. For each problem we present the dimension of the traffic matrix (n), the number of nonzero entries in the traffic matrix (m), the value of the optimal solution (z^*) found by the branch-and-bound algorithm proposed in Ref. 19, the value of the best solution ($GRASP$) found by algorithm GRASP_with_filter_for_TSA, an indication on whether this solution is optimal or not, the index of the

Table III. Run Statistics on Random Instances with the GRASP with Filter Algorithm

		GRASP_for_TSA				GRASP_with_filter_for_TSA			
Problem	n	Best	ITR	#_LS	Secs	Best	ITR	#_LS	Secs
R.01	15	6724	8	1000	377	6724	8	758	278
R.02	15	6771	1	1000	349	6771	1	1000	348
R.03	15	6475	693	1000	351	6475	693	997	350
R.04	15	6342	7	1000	385	6342	7	993	377
R.05	15	6415	626	1000	349	6415	626	989	350
R.06	15	7550	1	1000	412	7550	1	1000	411
R.07	15	6889	55	1000	430	6889	55	999	423
R.08	15	6076	586	1000	334	6078	219	994	328
R.09	15	6434	41	1000	336	6434	41	999	332
R.10	15	6525	5	1000	382	6525	5	994	375
Average	15	6620	202	1000	371	6620	166	972	357
R.11	18	7682	135	1000	914	7689	532	485	434
R.12	18	7693	580	1000	855	7698	235	973	811
R.13	18	7524	718	1000	957	7530	394	987	939
R.14	18	8254	65	1000	841	8254	65	1000	840
R.15	18	8021	1	1000	951	8021	1	1000	950
R.16	18	7755	30	1000	917	7755	30	1000	915
R.17	18	8032	222	1000	1017	8032	222	1000	1023
R.18	18	7234	156	1000	792	7246	169	455	377
R.19	18	6858	947	1000	720	6858	694	907	649
R.20	18	8027	183	1000	1054	8027	183	1000	1062
Average	18	7708	304	1000	902	7711	253	881	800
R.21	21	9287	769	1000	1821	9287	769	999	1823
R.22	21	8890	958	1000	1972	8890	958	1000	1974
R.23	21	9145	906	1000	2095	9145	906	1000	2075
R.24	21	8464	491	1000	1784	8464	491	1000	1741
R.25	21	8511	855	1000	1742	8511	855	1000	1700
R.26	21	8838	172	1000	2129	8838	172	1000	2081
R.27	21	7980	108	1000	1729	7980	108	1000	1694
R.28	21	9407	4	1000	2393	9407	4	1000	2335
R.29	21	8693	514	1000	1805	8693	514	999	1779
R.30	21	9816	45	1000	2398	9816	45	1000	2381
Average	21	8903	482	1000	1988	8903	482	1000	1958

iteration on which the best solution was found (itr), the number of times LocalSearch was effectively performed (#_*LS*), and the computation time in seconds (secs). These results show that the GRASP with filter algorithm was able to find the exact optimal solution for 23 of 36 instances of literature problems reported in this table. In the next section we present a new procedure that further improves these computational results.

4. Reactive GRASP

The restricted candidate list parameter α is basically the only parameter to be set in a practical implementation of a GRASP. Feo and Resende[6] have discussed the effect of the choice of the value of α in terms of solution quality and diversity during the construction phase and how it impacts the outcome of a GRASP procedure. In this section, we present a new procedure called *Reactive GRASP*, for which the restricted candidate list parameter α is self-adjusted according to the quality of the solutions previously found.

Instead of using a fixed value for the parameter α, which determines which elements will be placed in the restricted candidate list at each iteration of the construction phase, we propose to randomly select it from a discrete set $\mathcal{A} = \{\alpha_1, \ldots, \alpha_m\}$ containing m predetermined acceptable values. Using different values of α at different iterations allows for building different restricted candidate lists, eventually leading to the construction of different solutions which would never be built if a single, fixed value of α was used. For example, one may consider $\alpha_1 = 0.1$, $\alpha_2 = 0.2$, . . . , and $\alpha_{10} = 1$, as reported in Section 3 for the current application to the time slot assignment problem. Let p_i be the probability associated with the choice of α_i, for $i = 1, \ldots, m$. We take the initial values $p_i = 1/m$, $i = 1, \ldots, m$, corresponding to a uniform distribution.

Table IV. Run Statistics on Random Instances with GRASP_with_filter_for_TSA Using max_permutations = 2

		max_permutations = |L|				max_permutations = 2			
Problem	n	Best	ITR	#_LS	Secs	Best	ITR	#_LS	Secs
R.01	15	6724	8	758	278	6724	54	729	196
R.02	15	6771	1	1000	348	6771	1	1000	287
R.03	15	6475	693	997	350	6475	36	992	275
R.04	15	6342	7	993	377	6342	4	988	285
R.05	15	6415	626	989	350	6435	4	982	254
R.06	15	7550	1	1000	411	7550	1	1000	280
R.07	15	6889	55	999	423	6889	217	991	275
R.08	15	6078	219	994	328	6075	421	993	273
R.09	15	6434	41	999	332	6429	313	999	266
R.10	15	6525	5	994	375	6525	102	994	293
Average	15	6620	166	972	357	6622	149	967	268
R.11	18	7689	532	485	434	7681	340	475	145
R.12	18	7698	235	973	811	7696	387	948	569
R.13	18	7530	394	987	939	7530	423	983	660
R.14	18	8254	65	1000	840	8254	10	1000	665
R.15	18	8021	1	1000	950	8021	1	1000	685
R.16	18	7755	30	1000	915	7755	260	1000	688
R.17	18	8032	222	1000	1023	8033	19	999	709
R.18	18	7246	169	455	377	7257	101	482	218
R.19	18	6858	694	907	649	6866	982	827	365
R.20	18	8027	183	1000	1062	8027	83	1000	712
Average	18	7711	253	881	800	7712	261	871	542
R.21	21	9287	769	999	1823	9287	281	1000	1749
R.22	21	8890	958	1000	1974	8888	696	999	1828
R.23	21	9145	906	1000	2075	9145	316	1000	1970
R.24	21	8464	491	1000	1741	8464	319	1000	1627
R.25	21	8511	855	1000	1700	8502	14	1000	1586
R.26	21	8838	172	1000	2081	8840	392	1000	1751
R.27	21	7980	108	1000	1694	7980	72	995	1459
R.28	21	9407	4	1000	2335	9407	9	1000	2032
R.29	21	8693	514	999	1779	8693	367	999	1535
R.30	21	9816	45	1000	2381	9816	40	1000	2032
Average	21	8903	482	1000	1958	8902	251	999	1757

We propose to periodically update the probability distribution p_i, $i = 1, \ldots, m$, using information collected during the search. Different strategies for this update operation can be explored. We describe below an absolute qualification rule, based on the average value of the solutions obtained with each value of α, which is just one among such possible strategies. Recall that at each iteration some value of $\alpha = \alpha_i$ is randomly selected from $\mathcal{A}$ using the probability distribution p_i, $i = 1, \ldots, m$. At any GRASP iteration, let $F(S^*)$ be the value of the overall best solution already found. Moreover, let A_i be the average value of the solutions obtained taking $\alpha = \alpha_i$ in the construction phase. The probability distribution will be updated after the execution of each block of block_iterations iterations (we used block_iterations = 100 in our implementation). To do so, compute

$$q_i = \left(\frac{F(S^*)}{A_i}\right)^{\delta}$$

for all $i = 1, \ldots, m$, and obtain by normalization from the q_i the new values of the probabilities p_i, $i = 1, \ldots, m$, given by

$$p_i = q_i / \left(\sum_{j=1}^{j=m} q_j\right).$$

We notice that the most suitable some value of $\alpha = \alpha_i$ reveals itself to be (i.e., A_i is relatively smaller), the higher the associated value of q_i and, consequently, the higher the updated value of the probability p_i. Accordingly, in the next block of block_iterations iterations, the values of α that lead to better solutions will have higher probabilities and, consequently, will be more frequently used in the construction phase of the GRASP procedure. The exponent δ may be used and explored to differently atenuate the updated values of the probabilities p_i.

We call the above described approach *Reactive GRASP*

Table V. Run Statistics on Literature Problems with Algorithm GRASP_with_filter_for_TSA

Problem	n	m	z^*	*GRASP*	Optimal?	ITR	#_*LS*	Secs
P.06	5	16	565	565	yes	6	100	0.9
P.07	5	24	3771	3771	yes	1	1000	4.4
P.08	5	24	4049	4149	no	38	139	1.2
P.09	5	25	3388	3443	no	1	100	1.1
P.10	6	16	3983	3983	yes	1	1000	4.1
P.11	6	18	3380	3380	yes	1	1000	4.8
P.12	6	26	657	657	yes	5	605	3.7
P.13	6	34	3220	3245	no	1	964	9.2
P.14	7	31	3157	3157	yes	1	900	11.3
P.15	7	34	341	349	no	2	100	2.9
P.16	7	34	2343	2358	no	1	1000	15.4
P.17	7	34	3281	3359	no	4	132	3.6
P.18	7	37	3228	3228	yes	3	525	6.7
P.19	8	36	3710	3710	yes	1	1000	13.1
P.20	8	37	1830	1860	no	1	538	13.6
P.21	8	38	3660	3660	yes	6	386	10.4
P.22	8	38	1912	1925	no	7	1000	20.9
P.23	8	44	3770	3810	no	2	368	10.4
P.24	9	34	661	661	yes	1	1000	27.8
P.25	9	36	504	504	yes	1	647	17.8
P.26	9	37	520	520	yes	1	723	22.4
P.27	9	44	216	216	yes	152	100	7.0
P.28	10	44	1729	1729	yes	1	858	40.7
P.29	10	53	3470	3470	yes	1	1000	65.7
P.30	10	60	4891	4902	no	72	1000	54.3
P.31	11	47	620	620	yes	1	1000	70.9
P.32	11	51	2480	2480	yes	1	1000	72.9
P.33	11	56	3018	3018	yes	1	1000	61.5
P.34	12	74	1980	1980	yes	1	1000	133.3
P.35	12	86	2140	2140	yes	1	1000	134.7
P.36	12	101	7210	7210	yes	1	969	156.3
P.37	13	87	6360	6370	no	7	1000	222.3
P.38	13	94	2130	2130	yes	208	997	172.4
P.40	14	85	4879	4984	no	14	1000	222.9
P.41	14	116	2688	2688	yes	15	997	320.9
P.42	15	138	2466	2475	no	565	1000	402.0

because the value of the parameter α is not fixed but, instead, is self-adjusted according to the quality of the solutions found during the search. The pseudo-code of algorithm Reactive_GRASP implementing this scheme for the time slot assignment problem is outlined in Fig. 9.

We summarize in Table VI the computational results obtained with the *Reactive GRASP* approach for the same 36 literature instances, using $\delta = 10$ and max_iterations = 1000. For each problem, we present the value of the best solution (*R-GRASP*) found by the reactive approach implemented through algorithm Reactive_GRASP, an indication of whether this solution improves on the solution previously found by algorithm GRASP_with_filter_for_TSA, another indication of whether this solution is optimal or not, the index of the iteration on which the best solution was found (#_*iter*), the value ($\bar{\alpha}$) of α that led to the best solution found by algorithm Reactive_GRASP, and the computation time in seconds (secs).

These results show that the Reactive_GRASP algorithm was able to find the exact optimal solution for 7 of the 13 instances for which GRASP_with_filter_for_TSA failed. The value of exponent δ was fixed once and for all, and improved solutions were obtained without time-consuming calibrations involving the parameter α.

We also notice that, for most of the literature problems (18 of the 36 instances), the best solution found by algorithm Reactive_GRASP was obtained with $\bar{\alpha} = 0.1$, coinciding with the best value found for parameter α in Section 3.1. This result shows that the reactive procedure for updating the probability distribution can effectively be used to replace a time-consuming calibration process, with the additional advantage that it also gives chances to other values of α, which

```
procedure Reactive_GRASP(T,A = {α_1, ..., α_m},seed,max_iterations,
        max_permutations,max_eliminations,block_iterations,δ)
1   best_value ← ∞;
2   sum ← 0; times ← 0;
3   p_i ← 1/m, i = 1, ..., m;
4   sum_i ← 0, i = 1, ..., m;
5   n_i ← 0, i = 1, ..., m;
6   for k = 1, ..., max_iterations do
7       Randomly select α = α_i from A using probabilities p_i, i = 1, ..., m;
8       n_i ← n_i + 1;
9       S ← ConstructGreedyRandomizedSolution(T,α,seed);
10      if k ≥ 100                                          /* filter test */
11      then if 0.9 × (1 − β).F(S) > best_value
12           then do
13               S' ← S;
14               go to line 25;
15           end then;
16      S' ← LocalSearch(T,S,max_permutations,max_eliminations,
            seed);
17      if F(S') < best_value                          /* best solution update */
18      then do
19          S* ← S';
20          best_value ← F(S*);
21      end then;
22      times ← times + 1;                              /* filter computations */
23      sum ← sum + (F(S) − F(S'))/F(S);
24      β ← sum/times;
25      sum_i ← sum_i + F(S');                          /* reactive computations */
26      if mod(k, block_iterations) = 0
27      then do
28          A_i ← sum_i/n_i, i = 1, ..., m;
29          q_i ← (F(S*)/A_i)^δ, i = 1, ..., m;
30          p_i ← q_i/(Σ_{j=1}^{j=m} q_j), i = 1, ..., m;
31      end then;
32  end for;
33  return S*;
end Reactive_GRASP;
```

Figure 9. Pseudo-code of the *Reactive GRASP* procedure for time slot assignment.

in the present case allowed finding the best solution for several test problems.

5. Concluding Remarks

In this paper we described a GRASP for finding approximate solutions to the time slot assignment problem, arising in the context of the optimal operation of TDMA satellite systems. The first contribution of this work consists in the use of a new, original type of neighborhood during the local search phase, based on constructive and destructive moves. This neighborhood definition allows overcoming the difficulty of using more standard neighborhoods, such as exchange or insertion neighborhoods. The use of a filtering technique to bypass local search computations from unpromising solutions was another feature explored within this procedure. Extensive computational experiments on both randomly generated and literature problems illustrated the effectiveness of the proposed GRASP approach.

The major contribution of this paper consists in the development of a new variant of the GRASP approach, in which the parameter α which defines the restrictiveness of the candidate list is self-adjusted according to the quality of the solutions previously found. In the *Reactive GRASP* approach, instead of using a fixed value for the parameter α, we randomly select it from a discrete set of acceptable values. The associated probability distribution is periodically updated.

Table VI. Run Statistics on Literature Problems with Algorithm Reactive_GRASP

Problem	*R-GRASP*	Better?	Optimal?	#_*iter*	$\bar{\alpha}$	Secs
P.06	565	=	yes	7	0.5	1.1
P.07	3771	=	yes	2	0.6	4.0
P.08	4049	yes	yes	558	0.4	1.3
P.09	3388	yes	yes	155	0.4	3.0
P.10	3983	=	yes	1	0.1	4.5
P.11	3380	=	yes	1	0.1	4.7
P.12	657	=	yes	1	0.1	3.8
P.13	3230	yes	no	3	0.5	7.8
P.14	3157	=	yes	1	0.1	10.1
P.15	341	yes	yes	16	1.0	4.7
P.16	2343	yes	yes	31	0.6	14.5
P.17	3281	yes	yes	166	0.4	5.5
P.18	3228	=	yes	8	0.9	10.4
P.19	3710	=	yes	1	0.1	14.6
P.20	1860	=	no	1	0.1	20.0
P.21	3660	=	yes	7	0.6	18.4
P.22	1912	yes	yes	329	0.6	20.0
P.23	3810	=	no	1	0.1	21.4
P.24	661	=	yes	4	0.1	27.9
P.25	504	=	yes	8	0.5	23.9
P.26	520	=	yes	1	0.1	28.3
P.27	216	=	yes	40	0.8	7.8
P.28	1729	=	yes	1	0.1	44.5
P.29	3470	=	yes	1	0.1	65.7
P.30	4891	yes	yes	85	0.5	56.6
P.31	620	=	yes	1	0.1	70.9
P.32	2480	=	yes	1	0.1	70.9
P.33	3018	=	yes	2	0.5	62.3
P.34	1980	=	yes	4	0.9	131.9
P.35	2140	=	yes	3	0.1	135.0
P.36	7210	=	yes	1	0.1	163.1
P.37	6370	=	no	1	0.1	231.8
P.38	2130	=	yes	17	0.2	70.5
P.40	4984	=	no	2	0.6	224.2
P.41	2688	=	yes	1	0.1	313.7
P.42	2480	no	no	237	0.1	405.8

The experimental results on literature problems indicated that the new *Reactive GRASP* approach outperforms the basic GRASP algorithm for the time slot assignment problem, finding improved solutions for most of the problems for which the pure GRASP failed in finding the exact optimal solution. Since different values of α are used throughout the algorithm, this feature incorporates an additional diversification strategy to the basic GRASP. Moreover, the approach is robust and tuning is much more simple, in that the value of α is self-adjusted and no preliminary calibration efforts are needed.

Acknowledgments

The authors acknowledge N. Paciornik for some fruitful suggestions concerning the ideas underlying the *Reactive GRASP* proce-

dure. We are also grateful to an anonymous referee for constructive remarks.

References

1. E. BALAS and P.R. LANDWEER, 1983. Traffic Assignment in Communication Satellites, *Operations Research Letters 2*, 141–147.
2. G. BONGIOVANNI, D. COPPERSMITH, and C.K. WONG, 1981. An Optimum Time Slot Assignment Algorithm for an SS/TDMA System with Variable Number of Transponders, *IEEE Transactions on Communications COMM-29*, 721–726.
3. P.M. CAMERINI, F. MAFFIOLI, and G. TARTARA, 1981. Some Scheduling Algorithms for SS/TDMA Systems, *Proceedings of the 5th International Conference on Digital Satellite Communication*, 405–409.
4. M. DELL'AMICO, F. MAFFIOLI, and M. TRUBIAN, 1997. New Bounds for Optimum Traffic Assignment in Satellite Communication, Rapporto Interno 20.97, Politecnico di Milano, Dipartimento di Elettronica e Informazione, Milano, Italy.
5. G.D. DILL, 1977. TDMA, the State-of-the-Art, *Records of the IEEE Electronic and Aerospatial Systems Conference*, 31.5A–31.5H.
6. T.A. FEO and M.G. RESENDE, 1995. Greedy Randomized Adaptive Search Procedures, *Journal of Global Optimization 6*, 109–133.
7. A. GANZ and Y. GAO, 1992. Efficient Algorithms for SS/TDMA Scheduling, *IEEE Transactions on Communications 40*, 1367–1374.
8. I.S. GOPAL, 1982. *Scheduling Algorithms for Multi-Beam Communication Satellites*, Ph.D. Dissertation, Columbia University, New York.
9. I.S. GOPAL and C.K. WONG, 1985. Minimizing the Number of Switchings in an SS/TDMA System, *IEEE Transactions on Communications COMM-33*, 497–501.
10. T. INUKAI, 1978. Comments on "Analysis of a Switch Matrix for an SS/TDMA System," *Proceedings of the IEEE 66*, 1669–1670.
11. T. INUKAI, 1979. An Efficient SS/TDMA Time-Slot Assignment Algorithm, *IEEE Transactions on Communications 27*, 1449–1455.
12. Y. ITO, Y. URANO, T. MURATANI, and M. YAMAGUCHI, 1977. Analysis of a Switch Matrix for an SS/TDMA System, *Proceedings of the IEEE 65*, 411–419.
13. F. MAFFIOLI, 1977. Personal communication.
14. M. MINOUX, 1986. Optimal Traffic Assignment in an SS/TDMA Frame: A New Approach by Set Covering and Column Generation, *RAIRO Recherche Opérationnelle 20*, 1–13.
15. C.A. POMALAZA-RAEZ, 1988. A Note on Efficient SS/TDMA Assignment Algorithms, *IEEE Transactions on Communications 36*, 1078–1082.
16. F. RENDL, 1985. On the Complexity of Decomposing Matrices Arising in Satellite Communications, *Operations Research Letters 4*, 5–8.
17. F. RENDL, 1986. Personal communication.
18. M.G. RESENDE and C.C. RIBEIRO, 1997. A GRASP for Graph Planarization, *Networks 29*, 173–189.
19. C.C. RIBEIRO, M. MINOUX, and M.C. PENNA, 1989. An Optimal Column-Generation-with-Ranking Algorithm for Very Large Scale Set Partitioning Problems in Traffic Assignment, *European Journal of Operational Research 41*, 232–239.