ELSEVIER European Journal of Operational Research 132 (2001) 22–38

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Theory and Methodology

A Tabu search heuristic for the generalized assignment problem

Juan A. Díaz [1], Elena Fernández *

Dpt. d'Estadística i Investigació Operativa, Universitat Politècnica de Catalunya, Pau Gargallo 5, 08028 Barcelona, Spain

Received 18 August 1998; accepted 20 March 2000

Abstract

This paper considers the generalized assignment problem (GAP). It is a well-known *NP-hard* combinatorial optimization problem that is interesting in itself and also appears as a subproblem in other problems of practical importance. A Tabu search heuristic for the GAP is proposed. The algorithm uses recent and medium-term memory to dynamically adjust the weight of the penalty incurred for violating feasibility. The most distinctive features of the proposed algorithm are its simplicity and its flexibility. These two characteristics result in an algorithm that, compared to other well-known heuristic procedures, provides good quality solutions in competitive computational times. Computational experiments have been performed in order to evaluate the behavior of the proposed algorithm. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Generalized assignment problem; Metaheuristics; Tabu search; Local search

1. Introduction

The generalized assignment problem (GAP) is a well-known NP-hard combinatorial optimization problem. It considers a situation in which n jobs have to be processed by m agents. The agents have a capacity expressed in terms of a resource which is consumed by job processing. The decision maker seeks the minimum cost assignment of jobs to agents such that each job is assigned to exactly one agent subject to the agents' available capacity. Assignment of jobs to computers in a computer network, storage space allocation, design of communication networks with node capacity constraints, scheduling variable-length television commercials into time slots, etc., are examples of practical applications for the GAP. It also appears as a subproblem in many real-life problems such as vehicle routing, plant location, flexible manufacturing systems and resource scheduling.

In this paper, we propose a Tabu search algorithm for the GAP. Two distinctive features of this algorithm have to be remarked: its simplicity and flexibility. These two characteristics result in an algorithm that provides good quality solutions

* Corresponding author. Tel.: +34-93-4016948; fax: +34-93-4015855.

E-mail addresses: jadiaz@eio.upc.es (J.A. Díaz), elena@eio.upc.es (E. Fernández).

[1] Partially supported by grant 110598/110666 from Consejo Nacional de Ciencia y Tecnología (CONACYT), México.

that are competitive with other well-known heuristic procedures.

The flexibility is obtained from two sources: a strategic oscillation scheme that results from allowing the search to explore the infeasible solutions' space and an adaptively modified objective function. During the search, the solutions' space is enlarged to permit considering infeasible solutions. This gives the search a higher level of flexibility and some feasible solutions, that otherwise would not have been considered, can be reached by successively crossing the feasible/infeasible boundary. This strategy has already been used for the GAP by other authors (see [16,25–27]). As explained later on, the difference is that in our approach no solution is qualitatively preferred to others in terms of its structure so a priori all considered solutions are equally desired. This further increases the degree of flexibility of the procedure.

Our algorithm considers a modified objective function that includes a penalty term associated with infeasible solutions. The weight given to such penalty is dynamically adapted according to the history of the search. A similar idea has been proposed independently by Yagiura et al. [25], where the search parameter is also adjusted automatically according to the result of the previous iteration. However, in our approach the modified objective function plays a central role since it is the element that really takes care of guiding the search. This is achieved by using a combination of recency-based and medium-term memory for adjusting the penalty weight. The combination of these two memory components results in an objective function that is continuously adapting itself to the stage of the search. This is an issue that we believe is interesting since, traditionally memory has been widely used to explore complex neighborhood structures, whereas it has been used in a very limited fashion to obtain automated mechanisms to modify the objective function that guides the search process.

The adaptability of the proposed algorithm permits not having to resort to complex neighborhood structures. This, in its turn, has an important effect on the overall speed of the procedure. This is particularly true with the neighborhood structures that are explored (the classical shift and swap neighborhoods) and with the long-term memory component considered which is the same (the assignments' frequency) both for the intensification and diversification phases.

The result is an algorithm where the search strategy is based on (a) exploring large areas of the infeasible solutions' space and (b) dynamically adapting the objective function to the stage of the search.

The proposed algorithm has been tested on different sets of problems publicly available in Beasley OR-library (*http://mscmga.ms.ic.uk/jeb/orlib/gapinfo.html*) and in Yagiura library (*http://www-or.amp.i.kyoto-u.ac.jp/yagiura/gap*). It has also been compared to other well-known existing methods proposed by other authors in [7,16,21,25,27].

The results prove that the performance of the algorithm is remarkable since it produces high-quality solutions in small computational times.

This paper is organized as follows: in Section 2 a model for the GAP is given and a relaxation RGAP that will be used throughout the paper is presented. Section 3, reviews solution methods for the GAP found in the literature. The Tabu search heuristic for the GAP is explained in detail in Section 4: neighborhood structures, the neighbor generation mechanism, and the memory functions used are detailed in that section. Section 4 also presents the mechanism for dynamically adjusting the penalty parameter. In Section 5, computational results are presented and our approach is compared to other existing algorithms. Finally, Section 6 includes some conclusions and remarks.

2. The model

Let $I = \{1, \ldots, m\}$ be the set of agents and $J = \{1, \ldots, n\}$ the set of jobs. A standard integer programming formulation for the GAP is the following:

$$\text{(GAP)} \qquad \min \quad z = \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}, \tag{1}$$

s.t.

$$\sum_{i\in I} x_{ij} = 1 \quad \forall j \in J, \tag{2}$$

$$\sum_{j\in J} a_{ij}x_{ij} \leqslant b_i \quad \forall i \in I, \tag{3}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in I \quad \forall j \in J, \tag{4}$$

where c_{ij} is the assignment cost of job j to agent i, a_{ij} the resource required for processing job j by agent i, and b_i is the available capacity of agent i. Decision variables x_{ij} are set to 1 if job j is assigned to agent i, 0 otherwise. Constraints (2), together with the integrality conditions on the variables, state that each job is assigned to exactly one agent. Constraints (3) insure that the resource availabilities of agents are not exceeded.

In our approach, the assignment costs have been expressed in terms of their best assignment

$$\Delta_{ij} = c_{ij} - c\min_j,$$

where $c\min_j = \min\{c_{ij} : i \in I\}$ is the cost of the best assignment for job j. Δ_{ij} can be interpreted as the relative assignment cost of job j to agent i with respect to its best assignment cost. The use of the Δ_{ij}'s allows fair comparisons for deciding which jobs have to be assigned to a given agent, particularly when the assignment costs have different ranges for the various jobs. Then, the objective function can be rewritten as

$$\min z = K + \min\left\{\sum_{i\in I}\sum_{j\in J}\Delta_{ij}x_{ij}\right\},$$

where $K = \sum_{j\in J} c\min_j$, represents a fixed assignment cost which has to be incurred. Since K is constant we can remove it from (1) and consider the following equivalent objective function:

$$\min z' = \sum_{i\in I}\sum_{j\in J}\Delta_{ij}x_{ij}. \tag{1'}$$

Throughout the paper, we consider a relaxed formulation for the GAP in which the capacity constraints (3) are dropped and a penalty term is added to the objective function:

$$\text{(RGAP)} \qquad \min \quad z'' = \sum_{i\in I}\sum_{j\in J}\Delta_{ij}x_{ij} + P, \tag{1''}$$

$$\sum_{i\in I} x_{ij} = 1 \quad \forall j \in J, \tag{2}$$

$$x_{ij} \in \{0,1\} \quad \forall j \in J, \quad \forall i \in I. \tag{4}$$

The term P in the objective function evaluates the infeasibility of the solutions to RGAP, with respect to constraints (3) of GAP. It takes the form

$$P = \rho\sum_{i\in I} s_i,$$

where $s_i = \max\{\sum_{j\in J} a_{ij}x_{ij} - b_i, 0\}$ measures the amount by which the capacity of agent i is violated by a given solution. The parameter ρ is a penalty factor.

The reasons for considering RGAP are next explained. In the formulation of the GAP constraints (3) are 'difficult' and, usually, they are not easy to fulfill. For this reason these constraints may confine feasible solutions to a fairly narrow region. When exploring the neighbors of a given solution in a heuristic based method, restricting the search to feasible solutions using simple moves tends to generate very few trial solutions, forcing the process to terminate very quickly, often with low quality solutions. On the other hand, allowing some violation of feasibility has two advantages: first, it permits the execution of moves that are less complex than might otherwise be required. Second, it gives the search a higher level of flexibility and much larger areas of the solutions' space can be explored. The deviation from feasibility can be measured and controlled through a penalty term in the objective function. Violation of feasibility is allowed in several recent works for the GAP like the Tabu search approach considered in [16], the VDS proposed by Yagiura et al. [27], the VDS with branching search considered in [26] and the ejection chain approach proposed in [25]. However, these references, each one in its own context, follow the same a priori criterion that, in some sense, guides the search that is performed. Namely: among feasible solutions to RGAP, feasible solutions to GAP (those that fulfill constraints (3)) are always preferred. This is an important difference with respect to our approach. We consider all feasible solutions to RGAP equally attractive in terms of their structure. This is further explained in Section 4.2.

3. Related work

In this section, we present a brief review of solution methods found in literature for GAP. A comprehensive treatment of these methods can be found in [6] and in [21].

Exact methods for the GAP differ from each other in the way the lower bounds are computed. These bounds are derived from relaxations of the assignment constraints (constraints (2)), the knapsack constraints (constraints (3)) or the integrality conditions on the variables (constraints (4)). See [3,18,20,22–24].

Large-sized instances of the GAP have been tackled by means of approximate methods since exact methods have only proved to be effective for small-sized instances where the capacity constraints are loose. Different Lagrangean relaxations have been proposed in [2,9,13–15,17].

Other types of heuristic methods focused on providing good quality feasible solutions can be found in the literature for the GAP.

Martello and Toth [19] propose a two phase heuristic method MTH. In the first phase an initial solution is constructed using a greedy function to measure the 'desirability' of assigning job j to agent i. In the second phase the initial solution is improved using a simple local exchange procedure.

Cattrysse [4] considers a heuristic that incorporates a variable reduction procedure in order to reduce the size of the problem. This reduction procedure is based on the addition of valid inequalities (facets) of the knapsack polytopes associated to the capacity constraints. It starts solving the LP relaxation of the GAP. Then, violated valid inequalities for each knapsack constraint are identified and added to the formulation. This process continues until no further valid inequalities can be generated. Once the reduction step finishes, all variables with value 1 are fixed and the capacity of each agent adjusted. In a second step, Simulated Annealing is applied to solve the reduced problem.

Amini and Racer [1] developed a Variable Depth Search Heuristic (VDSH) for the GAP. Two phases are considered. In the first phase, an initial solution and the LP lower bound are generated. A doubly-nested iterative refinement procedure is performed in the second phase. Feasible shift moves of a job from one agent to another one, and feasible swap moves between jobs assigned to different agents are considered during the refinement procedure.

Trick [24] considers a heuristic approach based on the linear programming relaxation of the GAP. It is an iterative method which consists of three steps. In the first step all 'useless' variables are removed (a variable is considered useless if $a_{ij} > b_i$). In the second step the LP-relaxation is solved. During the third step all variables with value 1 are fixed. When the LP based heuristic finishes, an improvement phase that consists in swapping pairs of jobs is applied.

Cattrysse et al. [5] reformulate the GAP as a set partitioning problem. A heuristic procedure based on this reformulation is proposed. Each column represents a feasible assignment pattern of jobs to an agent. For each agent, columns are generated solving a knapsack problem in which the coefficients are obtained from the vector of dual variables of the LP-relaxation. Since this problem is degenerate, a dual ascent procedure is used to obtain the dual variables. Then, a subgradient optimization procedure is applied to improve the lower bound. Heuristic solutions may be found by coincidence during the subgradient optimization or by search among the columns generated.

Osman [21] proposes a hybrid heuristic SA/TS which combines Simulated Annealing and Tabu search. It uses a λ-generation mechanism which describes how a solution can be altered to generate another neighbor solution. The SA/TS algorithm is called hybrid because it combines the non-monotonic oscillation strategy of Tabu search with the simulating annealing philosophy. This non-monotonic cooling scheme is used within the framework of simulating annealing to induce an strategic oscillation behavior in the temperature values. Also a Tabu search algorithm is proposed for the GAP. Both, the hybrid SA/TS and the Tabu search algorithms use a frequency-based memory that records information used for diversification purposes.

A genetic algorithm (GA) for the GAP is proposed by Beasley and Chu in [7]. Instead of using a binary representation for the solutions, they use a n-dimensional vector of integer alphabets in the set I. These integer alphabets identify the agent to which each job is assigned. Fitness–unfitness pair evaluations are used to deal with both, feasible and infeasible solutions. Apart from using mutation and crossover operators, a two-phase heuristic improvement operator is also used: in the first phase this operator tries to recover feasibility by reducing the unfitness score. The second phase tries to improve the cost (fitness) of the solution without further violating the capacity constraints.

Laguna et al. [16] introduce a Tabu search algorithm for the multilevel assignment problem MGAP based on ejection chains. MGAP differs from GAP in that agents can perform tasks at different efficiency levels. An ejection chain is an embedded neighborhood construction that considers the neighborhoods of simple moves to create more complex and powerful moves. Multiple dynamic Tabu lists and a long-term memory component are used within the framework of Tabu search. Also a strategic oscillation scheme that allows searching paths to cross the capacity–feasibility boundary is used.

Yagiura et al. [27] have considered a Variable Depth Search algorithm VDS for the GAP. They incorporate an adaptive use of modified shift and swap neighborhoods where some moves are forbidden in order to avoid cycling. The method also allows the search to visit infeasible solutions modifying the objective function to penalize the violated capacity of the agents. In Yagiura et al. [26], a branching search processes to construct the neighborhoods are incorporated in order to improve the performance of the VDS algorithm [27].

Simultaneously to our work, Yagiura et al. [25] have proposed a Tabu search algorithm where ejection chains are used to create more complex and powerful moves. The algorithm maintains a balance between visits to feasible and infeasible regions using an automatic mechanism for adjusting the parameter that weights the penalty term in the objective function.

4. Tabu search

In this section, we present the components of our Tabu search algorithm for the GAP. First, the solution representation is described. Second, the neighborhood structures and their associated moves are presented. Then, a strategic oscillation scheme that permits the search to alternate between feasible and infeasible solutions is explained. Finally, the characteristics of the short-term memory and the frequency-based memory components of the algorithm are detailed.

Tabu search, introduced in Glover [11] and [12], exploits a collection of principles for intelligent problem solving. It uses: (a) flexible memory structures; (b) an associated mechanism of control, to be used along with the memory structures, to define Tabu restrictions and aspiration criteria and (c) records of historical information for different time spans that permit the use of strategies for intensification and diversification of the search process. Its power is based on the use of adaptive memory which is used for guiding the search process to overcome local optimality and to obtain near-optimal solutions. The use of short-term memory constitutes a form of aggressive search that tries to make good moves subject to the constraints implied by Tabu restrictions. Tabu restrictions are used to prevent cycling and aspiration criteria define rules to ignore Tabu restrictions under certain conditions. Intensification and diversification strategies are attained by means of long-term memory. Intensification strategies exploit features historically found desirable while diversification strategies encourage the search to explore unvisited regions.

4.1. Solutions

A solution π for both GAP and RGAP is represented with a vector of n elements, where the kth component is the agent to which job k is assigned, i.e.

$$\pi = (\pi_1, \ldots, \pi_n), \quad \text{where } \pi_j \in I; \; \pi_j = i \Longleftrightarrow x_{ij} = 1,$$

4.2. Neighborhood structures and generation mechanism

Given the set S of feasible solutions for an optimization problem, a neighborhood structure defines, for each solution $\pi \in S$, a set $S_\pi \subset S$ of solutions that in some sense are 'close' to π. We have considered two simple classical neighborhood structures: shift and swap. Given a solution π, the shift neighborhood, N_{shift}, is defined to be the set of solutions that can be obtained by reassigning one job from one agent to another (see Fig. 1).

$$N_{\text{shift}}(\pi) = \{(\pi'_1, \ldots, \pi'_n) | \exists j^* \in J \text{ s.t. } \pi'_{j^*} \neq \pi_{j^*}, \\ \pi'_j = \pi_j \ \forall j \neq j^*\}.$$

The swap neighborhood, N_{swap}, is the set of solutions that can be obtained by interchanging the assignments of two jobs, originally assigned to different agents (see Fig. 2).

$$N_{\text{swap}}(\pi) = \{(\pi'_1, \ldots, \pi' n) | \exists j_1,\ j_2 \in J, \pi_{j_1} \neq \pi_{j_2}, \\ \text{s.t. } \pi'_{j_1} = \pi_{j_2},\ \pi'_{j_2} = \pi_{j_1},\ \pi'_j = \pi_j \ \forall j \neq j_1, j_2\}.$$

It is important to note here that when only swap moves are used the solutions have a special structure. In particular, solutions have a fixed number of jobs assigned to each agent. On the other hand, shift moves are more restrictive in terms of feasibility. For these reasons we have used $N = N_{\text{shift}} \cup N_{\text{swap}}$ as neighborhood structure which is a more flexible option.

A neighbor generation mechanism is a set of rules for selecting a solution from a given neighborhood. The following rules define the generation mechanism used in our Tabu search implementation:

- Moves in $N_{\text{shift}} \cup N_{\text{swap}}$ that are not Tabu-active are considered admissible.
- For the current assignment $\pi = (\pi_1, \ldots, \pi_n)$, jobs are processed by decreasing values of $\Delta_{\pi_j j}$.
- For a given job j, shift and swap moves are

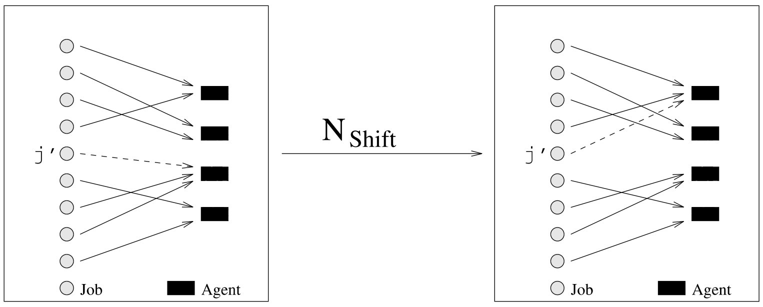

Fig. 1. Shift neighborhood.

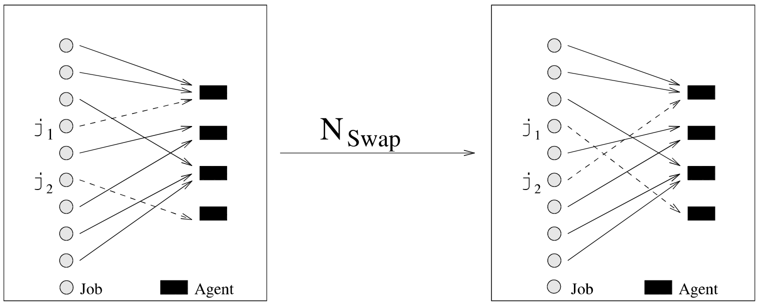

Fig. 2. Swap neighborhood.

evaluated and the best admissible move with respect to objective function ($1''$) is associated to the job.

- The candidate generation mechanism halts if the best admissible move for the job being processed improves the objective function value ($1''$). At each iteration, the candidate moves are regenerated.
- Otherwise, if all the jobs have been processed but none of the associated moves improve objective function ($1''$), the admissible move with the lowest increment is selected and performed.

4.2.1. Remarks

(1) Note that both, the considered neighborhood and the selection rules, are extremely simple. It is interesting to note that since the search is first improving, in most cases only a partial exploration of candidate moves is performed at each iteration. It is highly probable to find an improving move with respect to objective function ($1''$) before all jobs have been processed.

(2) As already mentioned, using the above strategy, the search is guided by the objective function ($1''$) and no priority is given to solutions that are feasible for the GAP. This does not happen in Laguna et al. [16], where the strategy of the search is guided towards feasibility through a qualitative criterion which makes prefer feasible solutions in most cases. Neither does this happen in Yagiura et al. [27], where the search strategy changes each time that feasibility is attained. In Yagiura et al. [25] the ejection chain moves do not consider solutions that increase infeasibility. Thus, their strategy also guides the search towards feasibility.

(3) It can also be observed that with the above selection rules, no priority is given to shift moves with respect to swap moves and vice versa. Once more the element that takes care of guiding the search is the modified objective function ($1''$). This allows the solutions obtained at the various iterations to change their structure and to have different numbers of jobs assigned to each agent. To a great extent this does not happen in [27] which uses a more complex neighborhood structure also based on $N_{\text{shift}} \cup N_{\text{swap}}$. In that work, at some given iterations a shift move is forced and then only swap moves are permitted until feasibility is recovered.

4.3. Strategic oscillation

As mentioned previously, the objective function used in the relaxed formulation RGAP includes a penalty to evaluate the violation of the agents' capacity constraints for the different solutions to RGAP. This penalty term is of the form

$$P = \rho \sum_{i \in I} s_i,$$

where ρ is a penalty parameter. Since the value of P is null for feasible solutions to the GAP, objective function ($1''$) in RGAP provides the original evaluation ($1'$) for such solutions and a modified evaluator for infeasible solutions. This modified evaluator, together with the (previously described) rules of movement guide the search towards the domain of feasibility when dealing with solutions far away from such domain. Therefore, considering objective function ($1''$) enables the algorithm to have a strategic oscillation behavior that permits alternating between feasible and infeasible solutions.

In this context, the weight assigned to the feasibility violation penalty, ρ, plays a central role to furnish the process with an equilibrium that permits alternating between feasibility and flexibility. In our algorithm this is done dynamically according to the expression

$$\rho := \rho \cdot \alpha^{(\text{infeas}/(\text{niter}-1)-1)},$$

where infeas is the number of infeasible solutions to GAP obtained during the last niter iterations and $\alpha \geqslant 1$ is a parameter that varies as explained later in this section.

The above expression can be interpreted in terms of incorporating a recency-based memory (infeas) and a medium-term memory (α) to adapt the parameter ρ to the current stage of the search.

The rationale behind the expression that we use is the following: the search strategy of our algorithm relies on the usefulness of a wider exploration of the infeasible solutions' space. For this reason, we consider that variations on the value of ρ, although necessary, should be performed very smoothly. This permits the search to operate according to a uniform criterion for a series of iterations. In other words, it allows the paths of successive solutions to be generated with respect to homogeneous rules. For a given α, the value of ρ ranges in $[\alpha^{-1} \cdot \rho, \alpha^{1/(\mathrm{niter}-1)} \cdot \rho]$. Note that, since the values of α will always be greater or equal to one, ρ will only increase when *all* the solutions found in the last niter iterations are infeasible. However, even in that case, the upper limit on the interval makes the increase in ρ to be very small. In the other cases, ρ is reduced (or maintained) increasing (or not decreasing) the importance assigned to the quality of the solutions versus their closeness to feasibility.

If, in the above expression, α were fixed to 1, the value of ρ would also be fixed during all the procedure and the history of the search would not be taken into account. When $\alpha = 2$, the expression obtained is very similar to the one used in [10] for Routing Problems with stochastic demands and customers. Observe, however, that when the value of α is fixed, only recency-based memory is used.

The value assigned to α influences considerably the progress of the search. The bigger the value of α the larger the range of variation for ρ. Therefore, when only infeasible solutions are visited in the last iterations, the rate of increase of ρ is faster for higher values of α. Since most of the time the considered solutions will be infeasible, small values of α tend to give smaller ρ's and vice versa. Consequently, a medium-term memory component has been incorporated to see if good feasible solutions have been found with a similar rate of increase/decrease for ρ. When this is so, the value of α is maintained to enhance the search under the same conditions on a given area. Otherwise, the value of α is increased guiding the search towards the feasible subregion of the area being explored. Again, the variation of α is extremely moderate to avoid sudden changes in the search. In particular, when the best feasible solution known so far has not been improved for the last 100 iterations, the value of α is increased by 0.005 every 10 iterations until a new better solution is found. Each time a new best feasible solution is found, the value of α is reset to the value 2.

4.4. Short-term memory phase

Fig. 3 outlines the short-term memory phase of the algorithm. The short-term memory component of Tabu search incorporates flexible memory functions to forbid moves in order to avoid previously visited solutions. This memory is often managed via solution's attributes that are forbidden during some period of time.

```
Short-term Memory Phase
short_term_phase($k, \pi^k$)
Let $\pi^*$ be the best solution found so far,
$\pi^k$ the current solution and $k$ the total number of trial solutions
generated
while (stopping_criterion not satisfied ) do
  begin
    Select a neighbor solution $\pi^{k+1} \in N(\pi^k)$
      (Using the neighborhood structure $N$ and the neighborhood
      generation mechanism detailed in the previous section)
    if ($cost(\pi^{k+1}) < cost(\pi^*)$) then
      $\pi^* := \pi^{k+1}$
    Update short-term memory
      (tabu restrictions detailed below)
    Update long-term memory
      (frequency-based memory detailed below)
    $k := k + 1$
  end
```

Fig. 3. Short-term memory phase.

The attributes we have considered are the assignments of jobs to agents. They are represented by ordered pairs (i, j) meaning that job j is assigned to agent i. In our implementation of Tabu search for the GAP some of these attributes are forbidden for the solutions to be generated in the next iterations. In other words, these attributes will be Tabu-active during a given number of iterations. More specifically, a matrix T is used to record the forbidden attributes for the future solutions. If during a given iteration k, a job j assigned to agent i_1 is reassigned, then, any solution having the pair (i_1, j) as attribute will be forbidden for the next t iterations. That is, $T_{ij} = k + t$, records the last iteration number for which the attribute (i_1, j) will remain Tabu-active. In the case of swap moves only one pair is recorded as Tabu-active in the matrix T. From the two assignments being considered it is the one with the higher $\Delta_{\pi_j j}$.

Recency-based memory is managed using dynamic Tabu tenures. The parameter t is randomly selected within a range $[t_{\min}, t_{\max}]$.

A standard aspiration criterion is used to bypass a Tabu restriction for a move. If, after performing the move, a feasible solution better than the best solution known so far would be obtained, the move is selected and performed.

As usual, the short-term memory phase is terminated after a fixed number of iterations without improvement.

4.5. Long-term memory phase

Long-term memory functions are used within the framework of Tabu search to furnish the search with a higher level of flexibility in order to achieve regional intensification and/or global diversification. The role of long-term memory is to record features of a selected number of trial solutions. Our TS algorithm uses a frequency-based memory scheme, both, for intensification and diversification purposes. The frequency matrix fr records the number of times that job j has been assigned to agent i in the total number of solutions visited up to that point.

The intensification phase recovers the best solution found so far and fixes the current assignments of jobs to agents with a frequency of at least 85%. This assignments can be considered ‘good’ given that: first, they are also present in the best solution found so far. Second, if they appear in most of the solutions generated so far either they have been selected many times or after being selected they have not been changed for a long period of time. Fixing some assignments is equivalent to reducing the size of the problem given that some decision variables are set to the value 1. Once this fixing process finishes, the short-term memory phase restarts for the resulting reduced GAP with the unfixed jobs and adjusted agents' capacities.

In the diversification phase, the frequency-based memory is also used but in a different way. The cost matrix Δ is replaced by the matrix $fr + \Delta$. Now the elements of the matrix fr can be viewed as penalties for selecting the most frequent assignments of jobs to agents. This is done to encourage the assignments that have seldom been selected and to discourage those assignments that have been frequently used. The matrix Δ is added to the frequency matrix for tie breaking purposes. During this phase it is intended to perform a number of high influence moves in order to produce solutions with different structures from the ones generated up to this point. After performing such moves, the original cost matrix Δ is recovered and the short-term memory phase reinitiated. Our experience has shown that this diversification scheme is effective since different high quality solutions from the ones generated to that point are usually obtained after a reduced number of iterations in the diversification phase.

Fig. 4 depicts the proposed Tabu search algorithm. It starts with one application of the short-term memory phase to improve the initial solution obtained using the MTH heuristic proposed by Martello and Toth [19]. Then, a cycle that alternates between intensification and diversification phases is performed during l iterations.

5. Computational results

Three different sets of problems have been used to test the proposed Tabu search algorithm. The first two sets of problems are publicly available in

```
Tabu Search Algorithm
Tabu_Search()
Let $\pi = (\pi_1, \dots, \pi_n)$ be the best solution found so far and $k$ the total
number of trial solutions generated
$k := 0$
Use $MTH$ (Martello & Toth heuristic) to obtain an initial solution $s_0$
Apply $short\_term\_phase(k, \pi^k)$
for $l$ iterations do
  begin
    {Intensification Phase}
      Retrieve $\pi^*$ and set $\pi^k = \pi^*$
      if $(fr_{\pi^*_j j} > 0.85 * k)$ then
        fix the value of $x_{\pi^*_j j}$ to 1
      Apply $short\_term\_phase(k, \pi^k)$
    {Diversification Phase}
      Unfix all fixed assignments
      Restart short-term memory phase for very few iterations using the
      matrix cost $\Delta + fr$
      Recover the original cost matrix $\Delta$
      Apply $short\_term\_phase(k, \pi^k)$
  end
```

Fig. 4. Tabu search algorithm for the GAP.

Beasley OR-library (*http://mscmga.ms.ic.ac.uk/jeb/orlib/gapinfo.html*). The third set of problems is publicly available in Yagiura library (*http://www-or.amp.i.kyoto-u.ac.jp/members/yagiura/gap/*).

The first set, S1, contains 60 'small-sized' maximization problems. These problems were used to test the set partitioning heuristic of Cattrysse et al. [5], the Simulated Annealing heuristic FSA of Cattrysse [4] as well as the hybrid Simulated Annealing/Tabu search heuristic SA/TS and the Tabu search heuristic TS proposed in Osman [21]. The test problems of this set have the following characteristics:

1. The number of agents m is set to 5, 8 and 10.
2. The ratio $r = m/n$ is set to 3, 4, 5 and 6 to determine the number of jobs n.
3. a_{ij} values are integers generated from a uniform distribution $U(5, 25)$.
4. c_{ij} values are integers generated from a uniform distribution $U(15, 25)$.
5. The b_i values are set to $((0.8/m)\sum_{j\in J} a_{ij})$.

The test problems are divided into 12 groups (each containing five problems), gap1 to gap12, according to the size of the test problems as is depicted in Table 1.

The second set of problems, S2, contains 24 'large-sized' minimization problems used to test the GA proposed in [7], the Variable Depth Search algorithm proposed in [27], the VDS with branching search considered in Yagiura et al. [26], and the ejection chain approach proposed in Yagiura et al. [25]. The third set of problems, S3, contains: 3 Type C problems with $n = 400$, 3 Type D problems with $n = 400$ and 9 Type E problems used in [25]. Dimensions of S2 and S3 problems can be seen in Table 5. The problems in S2 and S3 are divided into five classes according to the way they were generated:

1. Type A. a_{ij} are integers generated from an uniform distribution $U(5, 25)$, c_{ij} are integers

Table 1
Dimensions for S1 problems

Problem group	m	n
gap1	5	15
gap2	5	20
gap3	5	25
gap4	5	30
gap5	8	24
gap6	8	32
gap7	8	40
gap8	8	48
gap9	10	30
gap10	10	40
gap11	10	50
gap12	10	60

generated from an uniform distribution $U(10,50)$ and $b_i = 0.6 \times (n/m) \times 15 + 0.4R$, where

$$R = \max_{i\in I} \sum_{j\in J, I_j=i} a_{ij}$$

and

$$I_j = \min\{i | c_{ij} \leqslant c_{kj}\ \forall k \in I\}.$$

2. Type B. a_{ij} and c_{ij} generated as in Type A problems and b_i is set to 70% of the value given for Type A.
3. Type C. a_{ij} and c_{ij} generated as in Type A problems and $b_i = 0.8 \sum_{j\in J} a_{ij}/m$.
4. Type D. In this case a_{ij} are integers generated using an uniform distribution $U(1,100)$, $c_{ij} = 111 - a_{ij} + e$, where e are integers generated using an uniform distribution $U(-10,10)$ and $b_i = 0.8 \sum_{j\in J} a_{ij}/m$.
5. Type E.
   $a_{ij} = 1 - 10 \times \ln(\text{Uniform}(0,1])$,
   $c_{ij} = 1000/a_{ij} - 10 \times \text{Uniform}[0,1]$,
   $b_i = \max(0.8/m) \times \sum_{j=1}^{n}$,
   $a_{ij}, \max\{a_{ij}\ \forall j \in J\}$.

Type E problems were generated using the procedure detailed in Laguna et al. [16].

A fairly simplistic assumption with respect to the cost/resource relationship is assumed for Types A, B and C problems. However Types B and C problems are more difficult than Type A since the capacity constraints are tighter. Types D and E problems are known to be more difficult to solve. Results for Type A problems are not given since the MTH heuristic produces the optimal values in all cases.

The experiments have been performed on a Sun Sparc Station 10/30 with 4 HyperSparc at 100 MHz using only one processor. The algorithm is coded in C language. For each of the test problems 30 runs were performed. After some tuning the following parameters' values were used: number of intensification and diversification iterations, $l = 6$; the Tabu tenures values range from $t_{\min} = 2$ to $t_{\max} = 6$. The stopping criterion for the short-term memory phase has been adapted to the size of the test problems. In particular, for problems in S1, this parameter has been set to 350 iterations without improvement. For large-sized problems in S2 and S3, the stopping criterion has been fixed to 1500 iterations without improvement.

The value of α is initially set to 1. Starting with a smaller value of α has the effect of producing good quality feasible solutions at an earlier stage of the search. Once a feasible solution is found, the value of α is reset to the value 2 and it is permitted to take values between 2 and 3. Its value is reset to 2 every time a better solution is found. Arbitrarly, the value of ρ is initialized to 1.

For each of the 60 problems of S1 the following values were computed:

- Average percentage deviation from the optimal value, $\sigma_{\text{opt}_p} = \sum_{r=1}^{30} (O_p - S_{\text{pr}})/30$.
- Average best-solution time, $t_{\text{best}_p} = \sum_{r=1}^{30} t_{\text{best}_{\text{pr}}}/30$.
- Average execution time, $t_{\text{total}_p} = \sum_{r=1}^{30} t_{\text{total}_{\text{pr}}}/30$.

O_p denotes the optimal solution for problem p, and S_{pr} is the solution obtained for problem p during the rth execution. $t_{\text{best}_{\text{pr}}}$ and $t_{\text{total}_{\text{pr}}}$ denote the times needed to reach the best solution and the total running time for problem p during run r, respectively.

Additionally, the following values were calculated for each problem group (gap1 to gap12):

- Average percentage deviation from the optimal value, $\sigma_{\text{group}} = \sum_{p=1}^{5} \sigma_{\text{opt}_p}/5$.
- Average best-solution time, $t_{\text{best}_g} = \sum_{p=1}^{5} t_{\text{best}_p}/5$.
- Average execution time, $t_{\text{total}_g} = \sum_{p=1}^{5} t_{\text{total}_p}/5$.

Table 2 summarizes some characteristics of the methods used for comparison in our computational experiments. We have focused on the neighborhoods that are explored, the possibility of violating feasibility and the use of recent, medium-term and long-term memory.

Tables 3 and 4 show the results obtained with our Tabu search algorithm (denoted TS†) for problems in S1. Results are compared to the hybrid Simulated Annealing/Tabu search SA/TS and the Tabu search TS1 heuristics proposed in [21] and also to the GA proposed in Beasley and Chu [7]. For SA/TS and TS1, Table 3 shows the average percentage deviation from optimum of the best solution found for each problem, over all the problems in each group (see [21]). For the GA and the TS† approaches, this table shows the average percentage deviations from optimum (σ_{group}) taken over all the executions of the problems in each

Table 2
Algorithms' characteristics

		SA/TS	TS1	GA	BVDS	TS(MGAP)	TSEC	TS
Moves	Shift move	x	x	x	x	x	x	x
	Swap move	x	x	x	x	x	x	x
	Ejection chain					x	x	
Infeasibility	Allow search to visit infeasible solutions				x	x	x	x
	Moves towards feasibility are preferred				x	x	x	
Objective function	Automatic penalty adjustment						x	x
	Recency memory						1 it.	10 it
	Medium term memory							100 it
Long term memory	Diversification	x	x			x		x
	Intensification							x

Table 3
Average percentage deviation from optimal for S1

Problem set	SA/TS	TS1	GA	TS†
gap1	0.00	0.00	0.00	0.00
gap2	0.00	0.10	0.00	0.00
gap3	0.00	0.00	0.00	0.00
gap4	0.00	0.03	0.00	0.00
gap5	0.00	0.00	0.00	0.00
gap6	0.05	0.03	0.01	0.01
gap7	0.02	0.00	0.00	0.00
gap8	0.10	0.09	0.05	0.01
gap9	0.08	0.06	0.00	0.00
gap10	0.14	0.08	0.04	0.03
gap11	0.05	0.02	0.00	0.00
gap12	0.11	0.04	0.01	0.00
σ_{all}	0.210	0.070	0.009	0.004
σ_{best}	0.04	0.03	0.00	0.00
No. Optimal solutions	39	45	60	60

group. Also, for each of the approaches, the following averages over all groups of problems are depicted in Table 3:

- σ_{all}: Global average percentage deviation from optimum.
- σ_{best}: Average percentage deviation from optimum of the best solutions found.
- Number of problems for which the optimal solution was found at least once.

Table 4 shows the average best-solution time, t_{best}, and the average execution time, t_{total}, for each compared method and for each group of problems. Times are measured in seconds.

It can be seen that our Tabu search algorithm provides the optimal values for all the problems. In this sense it outperforms SA/TS and TS1 and is equivalent to GA. With respect to the deviation from optimal values it never exceeds 0.03% and, therefore, it outperforms all the compared methods.

Although the average best-solution times and average execution times are not comparable since the experiments were carried out in different computers, it must be noticed that for TS† the values t_{best} and t_{total} are extremely good (recall that we only use one processor).

We next show the results obtained for problems in S2 and S3. We compare our results to the best known solution for these problems. They are shown in the last column of Table 5. For five Type C instances, one Type D instance and six Type E instances this value corresponds to the optimal solution value obtained with the branch-and-bound algorithm proposed by Nauss [20]. These instances are marked with an asterisk. For the other instances, the best known values are not known to be optimal and have been taken from [8] in the case of Type B instances and from [25] for the remaining Types C, D and E instances.

Now, for each of the problems the following values were computed:

Table 4
CPU Times for S1[a]

Problem group	SA/TS[b]		TS1[b]		GA [c]		TS† [d]	
	t_{best}	t_{total}	t_{best}	t_{total}	t_{best}	t_{total}	t_{best}	t_{total}
gap1	n.a.	0.22	n.a.	0.73	0.40	72.38	0.04	1.09
gap2	n.a.	0.91	n.a.	1.44	1.68	79.96	0.04	1.48
gap3	n.a.	1.82	n.a.	2.85	2.18	85.06	0.08	2.05
gap4	n.a.	2.32	n.a.	4.86	4.16	92.36	0.13	2.54
gap5	n.a.	2.48	n.a.	3.97	5.52	100.36	0.22	2.38
gap6	n.a.	5.67	n.a.	8.85	16.36	130.02	0.41	3.53
gap7	n.a.	13.97	n.a.	12.72	10.54	130.34	0.35	4.92
gap8	n.a.	19.53	n.a.	22.99	49.90	184.42	2.38	7.64
gap9	n.a.	5.73	n.a.	12.19	13.82	129.32	0.77	3.72
gap10	n.a.	15.55	n.a.	18.62	33.70	167.66	1.07	5.90
gap11	n.a.	36.96	n.a.	34.46	32.60	193.00	1.53	8.02
gap12	n.a.	65.96	n.a.	47.07	39.08	260.34	1.83	10.67

[a] n.a. = Not available.
[b] CPU times in seconds on a VAX-8600 computer.
[c] CPU times in seconds on a Silicon Graphics Indigo R4000 (100 MHz).
[d] CPU times in seconds on a Sun Sparc Station 10/30 with 4 HyperSparc 100 MHz (SPECint95 2.35).

- Average percentage deviation from best known solution, $\sigma_{\text{best}_p} = \sum_{r=1}^{30} (S_{\text{pr}} - B_p)/30$.
- Average best-solution time, t_{best_r}.
- Average execution time, t_{total_p}.

B_p is the best known solution value for problem p.

Tables 5 and 6 show the results obtained. Now, TS† is compared to the GA proposed by Beasley and Chu in [7], to the Variable Depth Search with Branching algorithm BVDS proposed by Yagiura et al. [26], to the Tabu search algorithm for the MGAP, denoted TS(MAGAP), proposed by Laguna et al. [16], and to the Ejection Chain algorithm TSEC proposed by Yagiura et al. [25]. Results for BVDS and TS(MGAP) were taken from [25].

For each problem, Table 5 shows the best values (*best*) and the average percentage deviation from the best known solution (σ_{best}) found with the different approaches. Table 6 shows the average best-solution times, t_{best}, and the average execution times, t_{total}. Again, times are measured in seconds.

The results obtained confirm the validity of our approach. Type B instances were easily solved and the best known value was found for all these problems. For Type C problems our results are also remarkable: we obtained the best known solution in four instances and for the other five instances our best value is at most four units above the best known solution. The difficulty of problems of Type D is reflected in the quality of the results we obtained: for none of these problems we attained the best known value. However, for the instances with less than 400 jobs our solutions are of good quality, although the performance of our algorithm decreases for the very large instances. For Type E instances our best solution is very close to the best known value, excepting for the very large instances of 400 jobs where the quality of our solutions again tends to decrease. The values of the average deviations from the best known values confirm the good behavior of TS†. Comparing TS† with the other approaches we can see that for all types of instances our algorithm outperforms BVDS, TS(MGAP) and GA in terms of quality of the solutions and average deviation from the best known value (for the methods and instances that report these values). If we compare TS† with the Ejection Chain algorithm, the quality of our best solutions is nearly the same than those of TSEC for Type C problems and for problems with less than 400 jobs of Types D and E. However, for the very large Types D and E problems with 400 jobs, TS† shows a slight decrease in its performance as compared to TSEC. In terms of the average deviation from the best known value, the results of TSEC are extremely good and are

Table 5
Best values and average deviation from best values for S2, S3[a]

Prob.	m	n	BVDS	TS (MGAP)	GA		TSEC		TS†		Best known value
			Best	Best	Best	σ_{best}	Best	σ_{best}	Best	σ_{best}	
B	5	100	n.a.	n.a.	1843	0.347%	n.a.	n.a.	1843	0.000%	1843
B	10	100	n.a.	n.a.	1407	0.071%	n.a.	n.a.	1407	0.000%	1407
B	20	100	n.a.	n.a.	1166	0.069%	n.a.	n.a.	1166	0.097%	1166
B	5	200	n.a.	n.a.	3553	0.301%	n.a.	n.a.	3552	0.005%	3552
B	10	200	n.a.	n.a.	2831	0.308%	n.a.	n.a.	2828	0.046%	2828
B	20	200	n.a.	n.a.	2340	0.060%	n.a.	n.a.	2340	0.113%	2340
						0.193%				0.044%	
C	5	100	1931	1931	1931	0.378%	1931	0.000%	1931	0.000%	1931[b]
C	10	100	1402	1403	1403	0.285%	1402	0.000%	1402	0.043%	1402[b]
C	20	100	1244	1244	1244	0.515%	1243	0.000%	1243	0.284%	1243[b]
C	5	200	3456	3457	3458	0.229%	3456	0.000%	3457	0.034%	3456[b]
C	10	200	2809	2812	2814	0.481%	2806	0.007%	2807	0.105%	2806[b]
C	20	200	2401	2396	2397	0.619%	2391	0.025%	2391	0.139%	2391
C	10	400	5605	5608	n.a.	n.a.	5597	0.000%	5598	0.077%	5597
C	20	400	4795	4792	n.a.	n.a.	4783	0.021%	4786	0.201%	4782
C	40	400	4259	4251	n.a.	n.a.	4245	0.024%	4248	0.220%	4244
						0.418%		0.009%		0.123%	
D	5	100	6358	6386	6373	0.658%	6354	0.041%	6357	0.163%	6353[b]
D	10	100	6367	6398	6379	1.237%	6356	0.167%	6355	0.511%	6349
D	20	100	6275	6283	6269	1.652%	6215	0.387%	6220	0.905%	6196
D	5	200	12755	12788	12796	0.652%	12744	0.020%	12747	0.093%	12743
D	10	200	12480	12519	12601	1.536%	12445	0.076%	12457	0.277%	12436
D	20	200	12440	12416	12452	1.983%	12277	0.166%	12351	0.887%	12264
D	10	400	25032	25145	n.a.	n.a.	24976	0.017%	25039	0.284%	24974
D	20	400	24780	24872	n.a.	n.a.	24604	0.021%	24747	0.936%	24604
D	40	400	24724	24726	n.a.	n.a.	24460	0.043%	24707	1.441%	24456
						1.286%		0.104%		0.611%	
E	5	100	12681	12687	n.a.	n.a.	12681	0.003%	12681	0.040%	12681[b]
E	10	100	11585	11650	n.a.	n.a.	11577	0.000%	11581	0.087%	11577[b]
E	20	100	8499	8555	n.a.	n.a.	8439	0.055%	8460	0.703%	8436[b]
E	5	200	24942	25203	n.a.	n.a.	24930	0.000%	24931	0.027%	24930[b]
E	10	200	23346	23567	n.a.	n.a.	23307	0.004%	23318	0.116%	23307[b]
E	20	200	22475	22735	n.a.	n.a.	22379	0.022%	22422	0.446%	22379[b]
E	10	400	45878	172185	n.a.	n.a.	45746	0.003%	45781	0.077%	45746
E	20	400	45079	137153	n.a.	n.a.	44887	0.017%	45007	0.648%	44882
E	40	400	44898	63669	n.a.	n.a.	44596	0.065%	44921	1.266%	44579
								0.019%		0.379%	

[a] n.a. = Not available.
[b] Known to be optimal values.

clearly better although, in our opinion, our deviations are small.

The information of Table 6 permits to complete the analysis of our results. First of all, it should be noticed that the times correspond to experiments performed in different computers so a normalizing factor should be applied to obtain a fair comparison. In [25] Yagiura et al. give an estimate of 1 for the normalizing factor that should be applied to the times of BVDS, TS(MGAP) and TSEC and of 0.1 for the normalizing factor that should be applied to the times of GA. Taking into account the characteristics of the computer where we have performed our tests (SPECint95 of 2.35, 100 MHz

Table 6
CPU times for S2, S3[a]

Prob.	m	n	BVDS[b]	TS(MGAP)[a]	GA[c]		TS†[d]		TSEC[a]	
			t_{total}	t_{total}	t_{best}	t_{total}	t_{best}	t_{total}	t_{best}	t_{total}
B	5	100	n.a.	n.a.	126.9	288.2	n.a.	n.a.	17.5	95.8
B	10	100	n.a.	n.a.	30.1	276.0	n.a.	n.a.	10.6	97.2
B	20	100	n.a.	n.a.	191.5	617.3	n.a.	n.a.	47.5	160.0
B	5	200	n.a.	n.a.	439.5	790.0	n.a.	n.a.	70.5	339.1
B	10	200	n.a.	n.a.	608.4	1027.4	n.a.	n.a.	154.8	389.8
B	20	200	n.a.	n.a.	518.5	1323.5	n.a.	n.a.	222.7	465.4
C	5	100	150	150	139.1	302.4	0.6	150	3.0	83.6
C	10	100	150	150	170.6	394.2	3.0	150	28.7	103.3
C	20	100	150	150	279.9	669.3	21.6	150	66.9	130.7
C	5	200	300	300	531.2	810.1	3.7	300	79.5	316.1
C	10	200	300	300	628.6	1046.0	100.5	300	212.7	403.0
C	20	200	300	300	1095.9	1792.3	137.4	300	291.8	498.3
C	10	400	3000	3000	n.a.	n.a.	105.8	300	1348.8	1820.0
C	20	400	3000	3000	n.a.	n.a.	130.4	300	1713.3	2236.3
C	40	400	3000	3000	n.a.	n.a.	157.6	300	2333.2	3442.4
D	5	100	150	150	369.9	530.3	62.6	150	43.6	106.4
D	10	100	150	150	870.2	1094.7	107.2	150	83.2	133.4
D	20	100	150	150	1746.1	2126.1	111.0	150	119.5	167.4
D	5	200	300	300	1665.9	1942.8	95.5	300	226.8	363.2
D	10	200	300	300	2768.7	3189.6	129.2	300	205.0	397.4
D	20	200	300	300	4878.4	5565.1	120.7	300	348.2	515.6
D	10	400	3000	3000	n.a.	n.a.	16.1	300	1979.6	2310.0
D	20	400	3000	3000	n.a.	n.a.	81.3	300	2039.12	2440.1
D	40	400	3000	3000	n.a.	n.a.	165.2	300	2573.6	3129.3
E	5	100	150	20000	n.a.	n.a.	39.9	150	66.5	124.0
E	10	100	150	20000	n.a.	n.a.	31.6	150	88.3	148.2
E	20	100	150	20000	n.a.	n.a.	90.4	150	146.3	197.2
E	5	200	300	20000	n.a.	n.a.	20.0	300	204.7	351.4
E	10	200	300	20000	n.a.	n.a.	34.1	300	233.0	396.4
E	20	200	300	20000	n.a.	n.a.	209.3	300	425.2	585.3
E	10	400	3000	3000	n.a.	n.a.	260.7	300	492.5	891.1
E	20	400	3000	3000	n.a.	n.a	212.9	300	750.3	875.0
E	40	400	3000	3000	n.a.	n.a.	217.7	300	1035.5	1211.1

[a] n.a. = Not available.
[b] CPU times in seconds on a Sun Ultra 2 model 2300, 300 MHz (SPECint95 12.3) computer.
[c] CPU times in seconds on a Silicon Graphics Indigo R4000 (100 MHz) computer.
[d] CPU times in seconds on a Sun Sparc Station 10/30 with 4 HiperSparc at 100 MHz (SPECint95 2.35).

and 89 Mb) and comparing them with the characteristics of the computer of reference in [25] (SPECint95 of 12.3, 300 MHz and 1 Gb) we consider that the normalizing factor to be applied to our times should be at most 0.2. The times depicted in Table 6 are the actual running times without applying the normalizing factor. After considering this normalizing factor, it can be seen that our times are smaller than the times required by all the compared methods and instances, excepting for the 9 instances with 400 jobs where our times seem to be slightly higher than those of TSEC.

In our opinion the results of Tables 5 and 6 show the interest of our approach. For small, medium-size and large problems of dimensions until $m = 20$ and $n = 200$ we obtain solutions of a remarkable quality that either are optimal/best-known or are very close to them in very small computational times (taking into account the speed of the computer). For the very large instances, our approach is outperformed by TSEC.

However, we believe that the simplicity of our approach is an added value for our algorithm that makes it competitive with TSEC even for the larger instances. Anyhow, the obtained results confirm the success of the distinctive feature of our proposal that consists in using both recent and medium-term memory to automatically modify the objective function that guides the search.

6. Conclusions and remarks

In this paper the GAP is considered and a Tabu search heuristic approach is proposed. A relaxed formulation that allows the search to cross the capacity–feasibility boundary using a penalty term is considered. A strategic oscillation scheme is then used to permit alternating between feasible and infeasible solutions and to give the search a higher level of flexibility. Diversification and intensification strategies are also implemented by means of frequency-based memory. The resulting algorithm is very simple in terms of its basic components: the neighborhoods that are explored, the rules used to explore the neighborhoods and the use of recent and medium-term memory. Its distinctive feature is the use of recent and medium-term memory to guide the search via a modified objective function which is dynamically adapted. This is an issue that we believe is interesting since, traditionally memory has been widely used to explore complex neighborhood structures, whereas it has been used in a very limited fashion to obtain automated mechanisms to modify the objective function that guides the search process.

The performance of the algorithm has been evaluated with three different problem sets and compared to other existing algorithms.

The results obtained with the proposed algorithm are remarkable. In general, we obtain high quality solutions that either are optimal/best-known or are very close to them. However, for the very large instances, our approach is outperformed by one of the compared methods. In all the cases, the required computational times are excellent (taking into account the speed of the computers). In our opinion, the simplicity of our approach makes it competitive even for the larger instances. Moreover, we understand that these results could be significantly improved by exploring more complex neighborhoods and by considering more sophisticated rules to explore the neighborhoods. For this reason we believe that the obtained results fully justify the interest of our proposal of using short-term and medium-term memory to dynamically adapt the objective function for guiding the search in the infeasible solutions' space.

Acknowledgements

Authors are grateful to the anonymous referees for their valuable comments which have helped to improve this paper.

References

[1] M.M. Amini, M. Racer, A rigourous computational comparison of alternative solution methods for the generalized assignment problem, Management Science 40 (1994) 868–890.

[2] P. Barcia, K. Jörnsten, Improved Lagrangean decomposition: An application to the generalized assignment problem, European Journal of Operational Research 46 (1990) 84–92.

[3] J.F. Benders, J.A. van Nunen, A property of assignment type mixed linear programming problems, Operations Research Letters 2 (1983) 47–52.

[4] D.G. Cattrysse, Set Partitionig approaches to combinatorial optimization problems, PhD thesis, Centrum Industrieel Beleid, Katholieke Universiteit Leuven, Belgium, 1990.

[5] D.G. Cattrysse, M. Salomon, L.N. van Wassenhove, A set partitioning heuristic for the generalized assignment problem, European Journal of Operational Research 72 (1994) 167–174.

[6] D.G. Cattrysse, L.N. van Wassenhove, A survey of algorithms for the generalized assignment problem, European Journal of Operational Research 60 (1992) 260–272.

[7] P.C. Chu, J.E. Beasley, A genetic algorithm for the generalised assignment problem, Computers & Operation Research 24 (1997) 17–23.

[8] J.A. Díaz, E. Fernández, A Tabu search heuristic for the generalized assignment problem, Technical Report DR. 98/08, Departament d'Estadística i Investigació Operativa, Universitat Politècnica de Catalunya, 1998.

[9] M.L. Fisher, R. Jaikumar, L.N. van Wassenhove, A multiplier adjustment method for the generalised assignment problem, Management Science 32 (1986) 1095–1103.

[10] M. Gendreau, G. Laporte, R. Séguin, A Tabu search heuristic for the vehicle routing problem with stochastic

demands and customers, Operations Research 44 (1996) 469–477.

[11] F. Glover, Tabu search: Part I, ORSA Journal of Computing 1 (1989) 190–206.

[12] F. Glover, Tabu search: Part II, ORSA Journal of Computing 2 (1990) 4–32.

[13] M. Guignard, M.B. Rosenwein, An improved dual based algorithm for the generalized assignment problem, Operations Research 37 (1989) 658–663.

[14] K.O. Jörnsten, M. Näsberg, A new Lagrangean relaxation approach to the generalized assignment problem, European Journal of Operational Research 27 (1986) 313–323.

[15] T.D. Klastorin, An effective subgradient algorithm for the generalized assignment problem, Computers & Operations Research 6 (1979) 155–164.

[16] M. Laguna, J.P. Keely, J.L. González-Velarde, F. Glover, Tabu search for the multilevel generalized assignment problem, European Journal of Operational Research 82 (1995) 176–189.

[17] L.A.N. Lorena, M.G. Narciso, Relaxation heuristics for a generalized assignment problem, European Journal of Operational Research 91 (1996) 600–610.

[18] S. Martello, P. Toth, An algorithm for the generalized assignment problem, in: In Proceedings of the Ninth IFORS Conference, Hamburg, Germany, 1981.

[19] S. Martello, P. Toth, Knapsack problems: Algorithms and computer implementations, Wiley, Chichester, 1990.

[20] R.M. Nauss, Solving the classical generalized assignment problem, Technical Report, 1998.

[21] I.H. Osman, Heuristics for the generalised assignment problem: Simulated annealing and Tabu search approaches, OR Spectrum 17 (1995) 211–225.

[22] G.T. Ross, R.M. Soland, A branch and bound algorithm for the generalized assignment problem, Mathematical Programming 8 (1975) 91–103.

[23] M.W.P. Savelsbergh, A branch-and-price algorithm for the generalized assignment problem, Operations Research 45 (1997) 831–841.

[24] M.A. Trick, A linear relaxation heuristic for the generalized assignment problem, Naval Research Logistics 39 (1992) 137–151.

[25] M. Yagiura, T. Ibaraki, F. Glover, An ejection chain approach for the generalized assignment problem. Technical Report 99013, Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, 1999.

[26] M. Yagiura, T. Yamaguchi, T. Ibaraki, A variable depth search algorithm with branching search for the generalized assignment problem, Optimization Methods and Software 10 (1998) 419–441.

[27] M. Yagiura, T. Yamaguchi, T. Ibaraki, A variable depth search algorithm for the generalized assignment problem, in: S. Voss, S. Martello, I.H. Osman, C. Roucariol, (Eds.), Metha-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Dordrecht, 1999, pp. 459–471.