Heuristics for the generalised assignment problem: Simulated annealing and tabu search approaches

Article *in* OR Spectrum · January 1995

DOI: 10.1007/BF01720977

1 author:

Ibrahim H Osman
American University of Beirut
93 PUBLICATIONS **5,730** CITATIONS

SEE PROFILE

OR Spektrum (1995) 17: 211–225

OR Spektrum

Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches

Ibrahim H. Osman

Institute of Mathematics and Statistics, University of Kent, Canterbury, Kent CT2 7NF, UK (e-mail: I.H.Osman@ukc.ac.uk)

Received: 27 October 1993 / Accepted in revised form: 9 November 1994

Abstract. The generalised assignment problem (GAP) is the problem of finding a minimum cost assignment of a set of jobs to a set of agents. Each job is assigned to exactly one agent. The total demands of all jobs assigned to any agent can not exceed the total resources available to that agent. A review of exact and heuristic methods is presented. A λ-generation mechanism is introduced. Different search strategies and parameter settings are investigated for the λ-generation descent, hybrid simulated annealing/tabu search and tabu search heuristic methods. The developed methods incorporate a number of features that have proven useful for obtaining optimal and near optimal solutions. The effectiveness of our approaches is established by comparing their performance in terms of solution quality and computional requirement to other specialized branch-and-bound tree search, simulated annealing and set partitioning heuristics on a set of standard problems from the literature.

Zusammenfassung. Das verallgemeinerte Zuordnungsproblem (GAP) besteht darin, eine Menge von Aufträgen einer Menge von Agenten kostenminimal zuzuordnen. Jeder Auftrag wird genau einem Agenten zugeordnet; die Summe der Anforderungen der einem Agenten zugeordneten Aufträge ist durch die diesem zur Verfügung stehenden Ressourcen begrenzt. Die Arbeit gibt eine Übersicht über exakte und heuristische Lösungsverfahren zum GAP. Es wird ein λ-Generierungs-Mechanismus beschrieben, wobei verschiedene Suchstrategien (ein Hybridverfahren aus Simulated Annealing und Tabu Search sowie reine Tabu Search-Verfahren) sowie Parameterkonstellationen untersucht werden. Die entwickelten Methoden beinhalten eine Anzahl von Eigenschaften, die sich für die Erzielung von optimalen Lösungen sowie guten Näherungen als geeignet erwiesen haben. Die Effektivität der Ansätze wird über den Vergleich hinsichtlich Lösungsqualität und Berechnungsanforderungen mit anderen speziellen Verfahren wie Branch und Bound, Simulated Annealing sowie Partitionierungs-Heuristiken bei Anwendung auf Standardprobleme aus der Literatur gezeigt.

Key words: Generalised assignment problem, local search, simulated annealing, tabu search, heuristics, set partitioning, branch and bound

Schlüsselwörter: Verallgemeinertes Zuordnungsproblem, lokale Suche, simulated annealing, tabu search, set partitioning, branch und bound

1. Introduction

The generalised assignment problem (GAP) is a well-known optimization problem. It involves finding a minimum cost assignment of a set of jobs to a set of agents. Each job must be assigned to exactly one agent. The total resource required from any agent must not exceed its available capacity. The GAP is of a great practical and theoretical importance. Practically, it has many real-life applications in governments and industries. For instance, Ross and Soland [57] and later Klastorin [37] show how a class of public and private sectors facility location problems can be modelled as GAPs, Fisher and Jaikumar [17] and Gheysens, Golden and Assad [19] present GAP-based heuristics for the vehicle routing problem (VRP) and the VRP fleet mix, respectively. Zimokha and Rubinshtein [65] formulate a project plan for the research and development activities in research institutions as a generalised assignment problem. Shtub [59] shows that the formation of part families and technology cells in a manufacturing system is equivalent to the GAP. Similarly, Jänicke [31] uses the GAP formulation to solve the problem of assigning orders to appropriate subplants working in parallel in a computer-integrated manufacturing system. Lee [40] formulates a man-on-board automated storage/retrieval system as a variant of the generalized assignments model. Theoretically, the GAP has many extensions and variants (Gavish and Pirkul [18], Martello and Toth [42, 43], Mazzola [44], Mazzola and Neebe [45]).

The GAP is an NP-complete problem (see Fisher et al. [16]) so it is unlikely that an efficient exact method will

be found. Exact methods can only solve small-sized problems and practical-sized problems are often tackled by applying heuristics to obtain approximate solutions. Local search methods form a class of heuristics which include the well-known simulated annealing (SA) and tabu search (TS) for obtaining near optimal solutions. They have been applied to a variety of NP-complete problems. For general presentations of TS and SA principles with various successful applications to combinatorial optimization problems, the reader may refer to Van Laarhoven and Aarts [64], Collins, Eglese and Golden [11], Aarts and Korst [1], Osman [47], Osman and Laporte [50], Osman and Potts [51], Glover and Laguna [23], Glover, Taillard and de Werra [25] and the contributions in: Glover et al. [24], Reeves [55], Laporte and Osman [39] and Pesch and Voss [53].

This paper proposes new SA and TS metastrategy heuristics and investigates their performance on the GAP. The remaining of this paper is organised as follows. Firstly, the GAP is discussed and the relevant literature is reviewed. Secondly, we introduce a λ-generation mechanism to modify neighbourhood structures, investigate different selection and search strategies and descsribe local search implementations for descent, hybrid simulated annealing and tabu search and many variants of tabu search methods. Thirdly, computational results of the developed algorithms are compared with other results obtained from set partitioning, simulated annealing, branch-and-bound heuristics on bench-mark test problems from the literature. Finally, the paper is concluded with some remarks.

2. The generalized assignment problem

The GAP can be formulated as an integer linear problem as follows:

$$\text{Min} \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{1}$$

subject to:

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

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

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

where $I = \{1,\ldots m\}$ is the set of m agents, $J = \{1,\ldots n\}$ is the set of n jobs, c_{ij} is the cost of assigning job j to agent i, a_{ij} is the resource requirement for agent i to perform job j, b_i the available capacity of agent i. The decision variable x_{ij} equals 1 if agent i is to perform job j, 0 otherwise. The objective is to minimize the cost of the assignment (1) such that each job is assigned exactly to one agent (2) without exceeding the total resource capacity of each agent (3).

2.1. *Bound and exact methods*

Bounds are used in exact tree search methods to fathom nodes. They are usually derived from relaxations of either the assignment or the knapsack or the integer restriction constraints of the GAP formulation. These bound could be strengthened by adding surrogate constraints or valid inequalities to the relaxations.

Ross and Soland [56] use a relaxation in which the knapsack constraints (3) are removed to derive a lower bound. This bound is improved by adding minimum penalties which are computed by reassigning jobs from one agent to another in order to satisfy the removed constraints. Martello and Toth [41] present an equivalent formulation of the GAP with a maximized objective function. They obtain an upper bound for the equivalent formulation by removing the assignment constraints (2). The bound is further improved by computing a lower bound on the penalty to be paid in order to satisfy the violated constraints. Benders and Van Nunen [7] examine a linear programming (LP) relaxation of the GAP in which integrality constraints (4) are replaced with nonnegativity constraints. They show that the LP-relaxation bound tends to be strong when the number of jobs is large compared to the number of agents and when the resource capacities are rather loose. Trick [63] examines the basis structure of the LP-relaxation of the GAP and finds some nice theoretical properties. The LP-relaxation bound is improved by adding valid inequalities (cuts) to the LP formulation. The improvement of the "LP with cuts" bound over the LP-relaxation bound varies from 1.2% to 2.3% on average. The percentage of the optimal integer solutions over the "LP with cut" bounds varies between 2.1% and 10% on a problem set of size 20 jobs and 5 agents.

Fisher, Jaikumar and Van Wassenhove [16] develop a lower bound based on a Lagrangean relaxation obtained by dualizing the assignment constraints (2) into the objective function. They also develop a multiplier adjustment method for finding a good set of multipliers. Guinard and Rosenwein [28] elaborate on this relaxation and describe a framework to construct a stronger bound by a Lagrangean dual ascent procedure which exploits violations of assignment constraints not considered in Fisher et al. [16]. They add a surrogate constraint of the violated assignment constraints to the Lagrangean relaxed model. The surrogate constraint is then dualized into the objective function. Problems up to 50 customers and 10 agents are solved.

Jörnsten and Näsberg [32] propose a surrogate Lagrangean relaxation of the knapsack constraints (3). This relaxation produces multiple choice knapsack problems. They also introduce another approach based on a Lagrangean decomposition. In this approach, extra integer variables in the form of xy-constraints $y_{ij} = x_{ij}$ are added to the GAP formulation and (1) is multiplied by a constant ($\zeta + \eta = 1$) where ζ and η are two parameters. The xy-constraints are then dualized into the objective function. The resulting relaxation separates naturally into knapsack problems in the x-variables and another simple semi-assignment problem in the y-variables. The bound obtained through this method is stronger than the one obtained by relaxing either constraints (2) or constraints (3). Barcia and Jörnsten [5] present a Lagrangean approach combining the Lagrangean decomposition and bound improving sequences in Barcia and Holm [4]. They provide theoret-

ical and computational results about the sharpness of the bound obtained by the combined approach.

Facets are valid inequalities derived from violated constraints. Facets can be used to reduce the problem size or to strengthen the bounds. Jörnsten and Värbrand [34] combine a relaxation (surrogate or Lagrangean) strategy with a generation procedure of valid inequalities and discuss a reduction test procedure. They conclude that the combined surrogate procedure is more promising than the combined Lagrangean procedure. Gottlieb and Rao [27, 28] discuss different classes of valid inequalities with their theoretical properties.

Some of the above described bounding procedures are embedded in branch-and-bound tree search methods to obtain exact solutions. Among these are Ross and Soland [56], Fisher et al. [16], Martello and Toth [41, 42], Guinard and Rosenwein [28] and Jörnsten and Värbrand [34]. The largest-sized problems solved to optimally are of size 10 agents and 60 jobs by Cattrysse [8]. Recently, a branch-and-price algorithm for the GAP with a linear programming relaxation of a set partitioning formulation is proposed by Savelsbergh [58]. For further details, we refer to the two surveys by Cattrysse and Van Wassenhove [9] and Martello and Toth [43]. Finally, probabilistic analysis of the GAP is presented in Dyer and Frieze [14].

2.2. *Heuristic methods*

There are few heuristics for the GAP and the literature lacks a comprehensive review on such methods. In this section, we make a modest attempt to fill this gap.

(a) Martello and Toth (MTH). Martello and Toth [41] propose a heuristic for the GAP which can be described as follows. Let f_{ij} be a measure of the *desirability* of assigning job j to agent i. MTH iteratively considers all the unassigned jobs, and determines the job, j^*, which has the maximum difference between the smallest and the second smallest f_{ij}. The job j^* is then assigned to the agent for which f_{ij^*} is a minimum. In the second part of the heuristic, the current solution is improved by a simple local exchange procedure, which checks whether an improved solution can be found using a *shift procedure*. The shift procedure considers each job and tries to reassign it to the agent with the minimum f_{ij}. Four different f_{ij} functions are used: (a) $f_{ij}=c_{ij}$; (b) $f_{ij}=\frac{c_{ij}}{a_{ij}}$; (c) $f_{ij}=-a_{ij}$ and (d) $f_{ij}=-\frac{a_{ij}}{b_i}$. The best solution obtained by the choices (a)–(d) is considered as the MTH solution.

(b) Reduction with simulated annealing (FSA): Cattrysse [8] proposes a variable reduction procedure which fixes some variables x_{ij} to 1 in order to reduce the GAP problem into a smaller size. The reduction (Fixing) procedure is based on the addition of valid inequalities (facets) of the knapsack polytope. The Fixing procedure starts by solving the LP-relaxation of the GAP. Based on the solution variables, a violated valid inequality for every original knapsack constraint is added to the formulation. The extended formulation is then solved to generate more facets and the procedure is repeated until no further valid inequalities can be generated. At the end of the procedure, all integer variables, which have $x_{ij}=1$, are used to assign jobs to corresponding agents. The capacity of each agent is then adjusted, the fixed variables are removed and a reduced problem of smaller size is created. The reduced problem is solved by simulated annealing (SA) procedure and the combined Fixing/SA procedure is denoted FSA.

Cattrysse's simulated annealing uses a single vector V of dimension n to represent the GAP solution. Each entry indicates the assignment of job i to agent j, i.e. $V(i)=j$. Neighbouring solutions are randomly generated either by switching two jobs belonging to two different agents or shifting a job from one agent to another. The switch process is applied 70% while the shift process is used for the other 30%. Infeasible solutions are accepted with certain probabilities depending on their degree of infeasibility. The SA cooling schedule is a step-wise reduction scheme following the classification in Osman and Christofides [49], i.e., a number of iterations is performed at a given temperature value before it is reduced in a series of stages by the step-wise simulated annealing cooling schedule. On a set of relatively hard test problems with up to 60 jobs and 10 agents, the SA algorithm without the fixing procedure produces solutions which are on average within 3.90% from optimality. The combined FSA procedure produces better results which are within 0.72% on average from optimality with a huge saving in computation time.

Another SA scheme is also applied by Osman [46] and tested on the same test problems as FSA in Cattrysse [8]. Our SA scheme obtains solutions which are within 0.04% on average from optimality. This scheme will be presented as a contribution to this paper and will be explained later in the next section.

(c) Set Partitioning (SPH). Set partitioning heuristic (SPH) has been applied to the GAP by Cattrysse [8] and elaborated in Cattrysse, Salomon and Van Wassenhove [10]. In this approach, the GAP is formulated as a set partioning problem (SPP) in which a column represents a feasible assignment of a subset of jobs to a single agent. New columns are generated by solving for each agent a knapsack problem whose coefficients are obtained from the vector of the dual variables in the LP-relaxation of SPP. Since the LP-relaxation of SPP is degenerate, a dual ascent procedure is used to obtain the dual of the LP-relaxation. A subgradient optimization procedure is then applied to improve upon the lower bound. The heuristic solution may be found by coincidence during the column generation and the subgradient optimization procedures or by searching amongst the columns generated by enumeration. The enumeration procedure eliminates columns based on their reduced costs. On the same test problems used for the FSA heuristic, SPH obtains solutions which are on average within 0.13% from optimality. Combining the fixing procedure with SPH has improved the average to 0.09%.

(d) Incomplete optimization. A heuristic solution can be derived from any exact algorithm in several ways. One way is to use a partial exploration of the branch and bound

tree search or a cut-off time. Partial exploration can take place by using an invalid branching rule to eliminate branches on heuristic grounds. For instance, a node can be fathomed if the gap between the upper and lower bounds is within a certain percentage. A cut-off time could be implemented by obtaining an integral solution by depth-first search and then exploring the tree search for the remaining available time. We will implement the later approach using the Martello and Toth [42] (MTBB) and Fisher, Jaikumar and Van Wassenhove [17] (FJVBB) exact methods.

(e) Lagrangean relaxation. Klastorin [37] reports a two-phase Lagrangean relaxation heuristic which is obtained by dualizing the assignment constraints into the objective function. In the first phase, a modified subgradient procedure is used to find an optimal dual solution. This phase is terminated as soon as a *primal* feasible solution is found. In a second phase, a tree search is performed in the neighbourhood of the feasible solution in order to improve upon it. Computational tests show that the heuristic procedure yields solutions which are within an average deviation of 1% from optimality. Moreover, they conclude that the second phase is computationally expensive to provide any significant improvements.

Jörnsten and Näsberg [33] propose simple heuristics which modify the Lagrangean decomposition bounds to obtain feasible solutions. It was found that the best heuristic is the one which starts from a solution satisfying the semi-assignment constraints. If the capacity constraints are violated, another simple interchange procedure is used to recover feasibilities. The obtained feasible solution is further improved without destroying feasibility by reassigning within one or two semi-assignment constraints. Jörnsten and Värbrand [34] implement a pivot and complement heuristic developed by Balas and Martin [3] to find, if possible, a feasible solution for the Lagrangean Lower bound at each node of the tree search. A good feasible solution is hoped to be generated and the best solution found during the search is stored as an upper bound to reduce the tree search. No results are reported from the upper bound heuristic but only from the tree search on set of 5 test problems of 4 agents and 25 jobs.

(f) Neural networks. Fang and Li [15] make a modest attempt to solve the GAP by a neural networks approach. The neural networks consist of a matrix of $m \times n$ elements. The neuron at position (i, j) has either a value of 1 or 0, indicating whether job j is assigned to agent i or not. Equations of motion for the assignment and capacity constraints are formulated to transfer local information between the neurons in such a way neurons compete to become active. Computational results claimed to be within 0.08% from optimality on a set of random problems with 10 jobs and 3, 5, 7 or 10 agents. It is also reported that the neural networks outperform a slow cooling SA in some cases. It should be pointed out, that there is no explanation of the SA algorithm nor any mention to previous literature on the GAP despite the fact that exact algorithms for such small-sized problems do exist since early 1970's. We include this study not for its merit but only inviting more research in order to better judge the merit of the neural networks approach.

(g) Linear programming relaxation. Trick [63] proposes a linear programming (LP) relaxation of the GAP. A variable x_{ij} is defined to be "useless" if $a_{ij} > b_i$. The LP-heuristic iteratively uses the following three steps: (*i*) remove all useless variables, if no variables remain then stop, (*ii*) solve the LP relaxation; (*iii*) fix all variables which have values of 1, delete the jobs and update the agents capacities and return to step (*i*). An improvement procedure is applied at the end of the LP-heuristic. This procedure swaps two jobs on different agents or assigns a job to a different agent. It is shown that the LP-heuristic consistently provides better solutions than the MTH heuristic on five sets of large-sized problems varying from 20 to 500 jobs and 5 to 100 agents with different distributions. It was noticed that small-sized problems are more difficult to solve than the large-sized ones and problems of one type are more difficult to solve than any other types.

(h) Aggregate/disaggregate. Hallefjord, Jörnsten and Värbrand [29] propose an approximation algorithm in which the set of jobs is partitioned into groups using a hierarchical cluster analysis. The partition is used to define an aggregated GAP problem of size smaller than the original GAP. The optimal aggregated solution is then disaggregated in order to obtain a feasible solution to the original problem. A small problem is solved to illustrate the idea. Computational results are presented for only two problems: one with 4 agents and 25 jobs and the other with 4 agents and 1000 jobs.

(i) Local descent search. The second phase of MTH which considers each job and tries to reassign (or shift) it to the agent with the minimum f_{ij} is the simplest form of a local search procedure. Amini and Racer [2] and Racer and Amini [54] develop a variable-depth search heuristic (VDSH) which is two-phase local search descent method. Phase one develops an initial solution and a LP lower bound. Phase two consists of a doubly-nested iterative refinement process in which either a feasible shift of a job from one agent to another or a feasible swap of a job with another one from a different agent is performed. A comparison of VDSH with MTH in [42] is made on sets of problems with size varying from 50 to 200 jobs and 5 to 20 agents. It was shown that VDSH produces better results on average than MTH but the former required far more computational effort than the latter. The VDSH search and generation mechanism can be seen as special cases of the descent approaches in Osman [46, 47] with $\lambda = 1$.

To our knowledge, there were no attempts to solve the GAP by tabu search other than Osman [46, 47]. Laguna et al. [38], however, solve a related problem, namely the multilevel generalised assignment problem using a different tabu search implementation based on ejection chain method. In the next section, we present our implementations for the local search descent, hybrid SA/TS and TS methods.

3. Local search methods

A local search (LS) method starts with an initial solution, S. It iteratively attempts to improve upon the current

solution, S, by a series of local changes generated by a suitably defined neighbourhood mechanism until a stopping criterion is met. Since a local search method explores well-defined neighbourhoods looking for improved solutions, it is often called a neighbourhood search method and its final solution is called a local optimal solution. Any LS method consists of fundamental concepts: solution representation, construction of initial solutions, generation of neighbours, acceptance criteria and selection strategies of alternate solutions and a stopping criterion. We shall explain some of the common concepts and their choices that are used in our methods. Specific details are left for the implementation sections.

Solution representation

A feasible solution, S, for the GAP can be represented by a partition $S=\{S_i/i=1,\ldots,m\}$ where each S_i uniquely defines the set of jobs assigned to agent i. The aim is to find an optimal assignment (say S without loss of generality) that minimizes the total cost $C(S)$ and satisfies:

$$C(S)=\sum_{i\in I}\sum_{j\in S_i} c_{ij}$$

$$\bigcup_{i=1}^{m} S_i=\{1,\ldots,n\}, \qquad S_p\cap S_q=\emptyset,\ \forall p\neq q\in I;$$

$$\sum_{j\in S_i} a_{ij}\leq b_i, \qquad \forall i\in I=\{1,\ldots,m\}.$$

Initial solution and long term procedure

Initial solutions can be generated by any random procedure. In constraint optimization problems, however, it might be difficult to generate feasible solutions by such procedure and a constructive procedure is often used instead. Any constructive procedure would generate only one initial solution for a given problem instance. The objective of the Long Term (LT) procedure is to diversify the search by restarting the search from different regions. It enables the same construction procedure to generate different inital solutions using information collected during the previous search.

The LT procedure employs a memory function to collect frequency information on how long a given job remains assigned to an agent. The memory function takes the form of a $m\times n$ matrix, LTM, which is empty at the beginning of the search. When the best solution S_{best} is updated, the LTM is also updated as follows: $\mathrm{LTM}(i,j)=\mathrm{LTM}(i,j)+1\ \forall i\in I,\ \forall j\in S_i$. At the end of the LS stopping criterion, the matrix $\delta\times$LTM is added to the cost matrix C, where δ is a penalty parameter. It is possible to direct the search from the explored region by changing the value of δ, a good value of δ is set to 1. A new starting solution is constructed using the penalized cost matrix while the objective function value is calculated without penalties from the original cost matrix. The LT procedure continues the search until the LT stopping criterion is met when three restarts are attempted without improving the best LT solution. If the new starting solution is not different from the previously generated ones, then a random solution is attempted when possible otherwise the LT procedure is terminated. A similar approach using the LT strategy is used in Skorin-Kapov [60] to solve the quadratic assignment problem.

λ-generation mechanism

The generation mechanism describes how a solution S can be altered to generate another neighbouring solution S'. Here, we shall adopt for the GAP the λ-interchange generation (for simplicity λ-generation) mechanism that was introduced in Osman [47] for a class of combinatorial optimization problems. In the λ-generation mechanism, the set of neighbours S' of S is denoted by $N_\lambda(S)$, a move which transforms S into $S'\in N_\lambda(S)$ is denoted by λ-move, and the final local optimal solution is called a λ-opt solution.

The λ-generation mechanism has been successfully used in Osman and Christofides [49], with a special data structure in Osman [48], Osman and Salhi [52], Thangiah et al. [61], Thangiah, Osman and Sun [62]. This mechanism generalises the type of move used in Martello and Toth [41] that only shifts one job from one agent and reassigning it to another. The generalised procedure has a nice property that a λ-opt solution is also a μ-opt solution for any integer $\mu<\lambda$, see Osman and Christofides [49].

We next describe the λ-generation mechanism for the GAP: Given a feasible solution $S=\{S_i/\forall i=1,\ldots,m\}$ where S_i is the set of jobs assigned to agent i and a pair of sets (S_p, S_q) in S. A λ-move would replace any subset $\overline{S_p}\subseteq S_p$ of size $|\overline{S_p}|\leq\lambda$ with another subset $\overline{S_q}\subseteq S_q$ of size $|\overline{S_q}|\leq\lambda$. If the replacement is performed, the new sets of jobs become $S'_p=\{S_p-\overline{S_p}\}\cup\overline{S_q}$ and $S'_q=\{S_q-\overline{S_q}\}\cup\overline{S_p}$ resulting in a new neighbour, $S'=\{S-\{S_p\cup S_q\}\}\cup\{S'_p\cup S'_q\}$.

The λ-neighbourhood of S is the family of all possible solutions S' that can be reached from S in one λ-move, over all $m(m-1)/2$ combinations of pairs of sets in S. Each pair of sets is sequentially selected to generate λ-moves according to the following order without repetition

$$(S_{\sigma(1)}, S_{\sigma(2)}),\ldots(S_{\sigma(1)}, S_{\sigma(m)}), (S_{\sigma(2)}, S_{\sigma(3)}),\ldots,$$
$$(S_{\sigma(m-1)}, S_{\sigma(m)})$$

where σ is a permutation of agent indices (say, $\sigma(i)=i$, $\forall i\in I$). Given a selected pair of sets (S_p, S_q), all jobs in either sets are sequentially checked in a systematic order applying λ-moves to generate feasible solutions. We shall illustrate the type of λ-moves for the case for $\lambda=2$. Since we are dealing with ordered pairs of sets (S_p, S_q) with $p<q$, the sizes $(|\overline{S_p}|, |\overline{S_q}|)$ of possible subsets are defined by $\{(0,1), (0,2), (1,0), (1,1), (1,2), (2,0), (2,1)\ (2,2)\}$ and the subsets are generated by two types of 2-moves:

(*a*) A *shift* type generates subsets of sizes (0,1), (1,0), (0,2) and (2,0). In the case of (0,1), the λ-move would shift one job from S_q and reassign it to S_p whereas in the case of (1,0) a job from S_p would be shifted to S_q. Similarly in the case of (0,2) and (2,0) two jobs would be shifted from one agent to another.

(*b*) An *interchange* type generates subsets of sizes (1,1), (1,2), (2,1) and (2,2). For example, in the case of

(1,2) the 2-move would interchange one job from S_p with another subset of two jobs from S_q.

The size of the λ-neighbourhood grows with the value of λ. There are $m(m-1)/2$ pairs of agents, the jobs are searched sequentially for improved feasible solutions using the λ-generation mechanism. Let us assume that there are $\frac{n}{m}$ jobs on average assigned to each agent, the number of shift moves is then $\left(\frac{n}{m}\right)^{\lambda}$ while the number of interchange moves is $\left(\frac{n}{m}\right)^{2\lambda}$. Since each move requires a constant time, i.e. $O(1)$, then the λ-neighbourhood can be computed in $O\left(\frac{n^{2\lambda}}{m^{2(\lambda-1)}}\right)$. For $\lambda=1$, the neighbourhood size is computed in $O(n^2)$. Note that, the 2-moves would generate infeasible solutions but these solutions are not considered. A λ-move is called *admissible* if it leads to a feasible solution and also satisfies other admissibility conditions in the case of tabu search.

Acceptance criteria and selection strategies

The most common criterion is to accept solutions that improve upon the current solution. However other criteria will be discussed in the implementation section. In addition to defining the acceptance criteria and how to generate $N_\lambda(S)$, one must decide how to choose amongst the admissible solutions $S' \in N_\lambda(S)$. There are two section strategies:

(*i*) The *best-admissible* (BA) strategy examines the whole neighbourhood, $N_\lambda(S)$, and chooses the best solution in terms of admissibility and acceptance criteria. The size of the λ-neighbourhood is constant and a special data structure can be used to speed up the computation time of identifying the next best solution without recomputing the whole neighbourhood after each performed move, Osman [48].

(*ii*) The *first-admissible* (FA) strategy follows a less aggressive orientation. It selects the first solution $S' \in N_\lambda(S)$ that satisfies the admissibility and acceptance criteria. The λ-neighbourhood is of variable sizes. After a move is performed the search continues from the current position to the end of the neighbourhood and back to the current position searching in a circular list.

Step 1. Initialization: Generate an initial solution S for either a local search (LS) or a long term (LT) procedure by:
- (*i*) **Either** the MTH construction procedure.
- (*ii*) **Or** a random procedure.

Let the best LS solution S_{best} take the configuration of S.

Step 2. Selection and acceptance criterion:
- (*i*) Select $S' \in N_\lambda(S)$ *sequentially* according to the first-admissible (FA) or the best-admissible (BA) selection strategies.
- (*ii*) **If** S' is accepted by the defined acceptance criterion, go to Step 3,
  Else go to Step 4.

Step 3. Update parameters for specific LS methods:
- (*i*) The current solution by setting $S=S'$.
- (*ii*) The best LS solution, if necessary, i.e., $S_{best}=S$.
- (*iii*) LT and short term memory functions for descent and TS.
- (*iv*) SA cooling schedule parameters.

Step 4. LS Stopping Criterion
- (*i*) **If** the LS stopping criterion is not met, go to Step 2.
- (*ii*) **If** the optimal solution is found, go to Step 6.

Step 5. LT Stopping Criterion
- (*i*) **If** the LT strategy is not required, go to Step 6.
- (*ii*) **If** the LT stopping criterion is not met, **Then**
  - update the LT best solution, S''_{best}, if necessary,
  - penalize the original cost matrix using the LT memory function and go to Step 1.

  Else go to Step 6.

Step 6. Stop with a λ-*opt* solution either S_{best} by the LS procedure, or S''_{best}, by the LT procedure.

Fig. 1. A general framework for our local search method

A general framework for the local search methods

After an explanation of the basis concepts, were now present a general framework for the local search methods in Fig. 1.

Local search methods differ in the way choices are made at each step of the above general framework. These choices are explained next.

3.1. Descent implementation

The local search (LS) descent method starts with a MTH solution or a LT solution. The LS descent method selects a neighbouring solution $S' \in N_\lambda(S)$ using the λ-moves according to the FA or BA selection strategies. It then evaluates the objective function values of $C(S)$, $C(S')$ and $\Delta = C(S') - C(S)$. If $\Delta < 0$, S' is accepted to replace the currnet solution for the next iteration. Alternatively, if $\Delta \geq 0$, S is retained. The search usually continues until the LS stopping criterion is met (the whole λ-neighbourhood of the current solution is searched without finding any improvement). At this point, the algorithm stops and S is declared as the λ-opt solution. If the LT strategy is applied, the LS descent is restarted from a different initial solution and continues until the LT stopping criterion is met.

3.2. Simulated annealing implementation

Simulated Annealing (SA) has its origin in statistical mechanics. The interest in SA began with the work of Kirkpatrick et al. [35] who propose a SA algorithm based on the analogy between the annealing of solids and the problem of solving combinatorial optimization problems. SA algorithm is a metastrategy LS method which attempts to avoid producing a poor local optimum inherent in a LS de-

scent method, by employing random selection and acceptance strategies. The random acceptance strategy allows "uphill" moves to be occasionally accepted with certain probabilities. These probabilites are determined by a control parameter (T) called "temperature" which is updated according to a deterministic "cooling" schedule. "Downhill" moves are always accepted like in the LS descent method. We adopt for the GAP the non-monotonic cooling introduced in Osman [47]. After each iteration, the non-monotonic cooling schedule controls the normal temperature decreases, the occasional temperature increases after the occurrence of a special neighbourhood without accepting any 1-move. These rules with the SA parameters settings are explained as follows.

(i) Initial and final temperature values. The initial temperature, T_s, is given the largest change in the objective function values, δ_{max}. The final T_f is assigned the smallest change in the objective function value, δ_{min}. The T_f parameter is no longer used to detect the algorithm stopping as our cooling schedule uses a different stopping criterion.

(ii) Temperature update rule. After each iteration k is performed, the temperatures are updated according to the following rules:

***Either** a normal decrement rule:*

$$T_{k+1} = \frac{T_k}{(1+\beta_k T_k)} \text{ where } \beta_k = \frac{T_s - T_f}{(\alpha+\gamma\sqrt{k})T_s T_f}.$$

***Or** an occasional increment rule:*

$$T_{reset} = \max\left\{\frac{T_{reset}}{2}, T_{found}\right\} \text{and } T_{k+1} = T_{reset}.$$

If the whole 1-neighbourhood is searched without accepting any solutions, we say the incremental condition is satisfied. The current temperature is then reset to a higher value, T_{reset} or to the temperature T_{found} at which the best solution was found. The reason is to escape from the current 1-opt solution and to allow for some further moves to be made but not to deviate much from the current solution.

The aim of the update rules is to provide fast reductions in the temperature values at the beginning of the search, then generate a sea-wave pattern in the middle of the search and ideal slower reductions near the end. A typical temperature trajectory is depicted in Fig. 2. The choice of α and γ values and the updating rule for T_{reset} would affect the running time and the quality of solution. The values of the cooling schedule parameters are determined experimentally by scanning the 1-neighbourhood of the initial solution to find the values for δ_{max}, δ_{min} and *Nfeas* the number of feasible 1-moves in the neighbourhood. Good results are obtained with $\alpha = n \times Nfeas$ and $\gamma = n$. T_{reset} is initially set to δ_{max} and subsequently divided by a half whereas T_{found} is given the temperature value at which the current best solution is found. These settings are based on experimentations with this tpye of neighbourhood. However, other mechanisms can be derived to control the temperature oscillation.

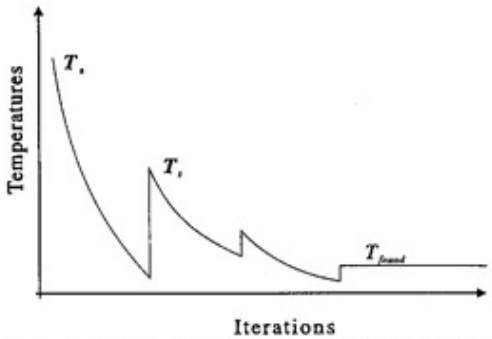

Fig. 2. Temperature changes in the non-monotonic cooling schedule

(iii) Stopping criterion. The SA stopping criterion can be arbitrarily chosen to reflect the amount of time one wants to spend on solving the problem. The stopping criterion we used is based on the number of temperature resets, *NR*, to be performed after the best solution is found.

The design of our cooling schedule to induce an oscillation behaviour in the temperature values as a mechanism for controlling search through variation in the objective function is similar to the strategic oscillation approach of tabu search in Glover [20, 21]. The manipulation of parameter thresholds to produce strategic oscillation in the tabu search framework is typically supported by memory structures to guide the search to guide its non-monotonic behaviour and to fulfil associated functions of controlling the search. Our SA implementation combines the non-monotonic oscillation strategy of tabu search with simulated annealing philosophy without much use of memory except for keeping the best known solutions and the temperature reset values. Hence our SA algorithm can take the name of a hybrid SA/TS algorithm or more specifically a hybrid "stimulated annealing/oscillation strategy" algorithm.

The success of our non-monotonic oscillating approach suggests the possibility of combining it with other forms of probabilistic guidance (as in probabilistic tabu search for example) as an alternative to accept/reject choice of SA and the deterministic memory structure of most TS. It could also be used in combination with other types of SA cooling schedules. The hybrid SA/TS approach has been successful in outperforming other methods in the literature for a number of combinatorial optimization problems such as, the vehicle routing problems [48, 52, 61, 62], the capacitated clustering problem [49], the maximal planar graph [30]. More details on the above applications and others can be found in Osman [47].

(iv) Algorithm description. Our SA/TS algorithm starts with an initial solution, S, and uses the 1-generation mechanism. The SA/TS algorithm selects a neighbour $S' \in N_1(S)$ in a *sequential order* strategy rather than the usual *random* strategy. At a given iteration k, one 1-move is generated and a decision is made to whether accept or reject such a

move. If the 1-move results in an improvement, $\Delta \leq 0$, then S' is accepted to replace S as the current solution for the next iteration. If $\Delta > 0$, then this non-improving move is accepted with probability $e^{-\frac{\Delta}{T_k}}$ where T_k is a positive value for T at iteration k, otherwise S is retained. According to the decision made, the current and best solutions are updated, if necessary, the temperature value is then updated for the next iteration $k+1$. The SA/TS algorithm is terminated when the SA/TS stopping criterion is met, otherwise a new iteration is started. If the LT strategy is applied and the LT stopping criterion is not met, the algorithm is restarted for another run.

3.3. Tabu search implementation

Tabu search (TS) is a procedure for guiding LS methods to overcome local optimality and obtain near-optimal solutions for hard combinatorial optimization problems. TS is introduced by Glover [20] and uses ideas from artificial intelligence. TS accepts a move to the best neighbour irrespective of whether it is a superior solution. Memory structures are used to forbid certain moves thus preventing the method from cycling. Different search strategies and heuristic rules are also used to explore the solution space. Since tabu search uses various heuristics, strategies and memory structures it is often called a "metastrategy" algorithm or a "metaheuristic".

TS operates by incorporating flexible memory functions to forbid moves that reinstates certain attributes of past accepted solutions. Attributes that are not permitted to be reinstated are called *tabu* and maintained in a short term memory on a list called *tabu-list*. After a number of iterations, called *tabu-list size*, attributes on the list are removed and are permitted to be reinstated. *Tabu restrictions* are based on attributes on the tabu list. A move is called *tabu-move* if its attributes satisfy tabu restrictions. *Aspiration criterion* is a test to correct wrongly diagnosed tabu moves. A λ-move is called *admissible* if it is feasible and not a tabu move or it has passed the aspiration test. A recent trend in tabu search is the use of intermediate and long term memory to intensify and to diversify the search into regions not yet visited.

A dynamic approach to manage the tabu list is successfully implemented in Dammeyer and Voss [12]. Statistical estimate of tabu list sizes and variable length tabu restrictions are also used in Osman [48]. Further details on the above ideas can be found in Glover [21, 22], De Werra and Hertz [13] and Glover and Laguna [23]. In this section, we will explain the TS components and strategies that are used within our implementation.

(i) Tabu list. Let us define a "zero" job involved in the shift moves by a dummy job with a zero cost and zero requirements. Let the tabu-list, *TABL*, take the form of a matrix of dimensions $(n+1) \times m$ with $n+1$ rows (one row for each job including the zero job) and m columns (one column for each agent). A 1-move from S to S' is applied to a given pair of sets (S_p, S_q) of S. The 1-move either shifts a job $e_1 \in S_p$ or exchanges it with another job $e_2 \in S_q$ at a given iteration k. Then, after such a 1-move is performed, the entries TABL (i_1, p) and TABL (i_2, q) are given the value, $k+|T_k|$, which is the future iteration number at which jobs e_1 and e_2 are allowed to return to S_p and S_q, respectively. Note that, i_1 and i_2 are the indices of e_1 and e_2 respectively in the set of $(n+1)$ jobs including the "zero" job.

(ii) Tabu restrictions. Recorded attributes are used to impose restrictions that prevent potential 1-moves from being reversed. We have implemented two kinds of tabu restrictions. A 1-move is considered tabu if its attributes satisfy:

(R1) e_1 returns to S_p ***and*** e_2 returns to S_q.
(R2) ***either*** e_1 returns to S_p ***or*** e_2 returns to S_q.

Restriction R2 is more restrictive than R1 since it makes more moves tabu. The tabu status of a move at iteration, say t, can be easily identified using a simple test. For example, when R1 is used, a move is tabu if:

$t \leq TABL(i_1, p)$ and $t \leq TABL(i_2, q)$

(iii) Aspiration criteria. Aspiration criteria are used to determine when tabu restrictions can be overridden, i.e., allow a tabu move to be performed if selected. Two aspiration criteria are implemented. The motive is to demonstrate how different aspiration criteria can influence the performance of tabu search. A tabu move from S to $S' \in N_1(S)$ is executed if it satisfies:
(A1) the objective function value, $C(S')$ is better than $C(S_{best})$ the objective function value of the best found so far.
(A2) the objective function values, $C(S')$ is better than the minimum of $\{ASP(e_1)$ and $ASP(e_2)\}$, where $ASP(e_i)$ is the current objective function value at the time job e_j is recorded in the tabu list.

(iv) Stopping criterion. Tabu search algorithm is terminated after a maximum number of iterations, *MAXI*, is performed without improving the best solution found so far.

(v) Algorithm description. The TS algorithm starts from an initial solution S. At each iteration, TS searches sequentially all admissible solutions $S' \in N_1(S)$ and selects one neighbour to replace the current solution S according to an acceptance criterion. The acceptance criterion chooses the best admissible move which has the most improvement or the least-improvement in the objective function value if the BA strategy is used. However, if the FA selection strategy is used, the acceptance criterion selects the first admissible solution that improves immediately upon the current objective function and the search continues from the new accepted solution. If there is no improving solution available, the least non-improving move in the neighbourhood is then performed. After performing the accepted move, its attributes are recorded in the tabu list. The TS algorithm is then continued until the TS termination criterion is satisfied. If the LT strategy is applied and the LT stopping criterion is not met, the TS algorithm is restarted for another run.

Note that, the FA selection strategy chooses the least non-improving move if there is no improving solution at any iteration. In this case, FA selects the same move that would have been selected by the BA strategy in order to

perform a mild diversification. The trajectory of solutions with the FA strategy is different from that of BA strategy due to variable length neighbourhood sizes. In regions where there are many improving moves in the neighbourhood, the FA strategy would select the first move, then update the tabu list, next select the first improving move subject to the newly accepted solution which can be different from the BA solution, etc. This strategy would then provide a search intensification and less restrictions since the tabu list is updated more frequently. Furthermore, the FA selection strategy can be seen as a combined strategy of FA and BA strategies and has an advantage when the size of the neighbourhood is large.

4. Computational experience

4.1. Test problems

We have investigated the performance of the developed algorithms on a set of 60 test problems from the literature. These problems have the following characteristics:

- the number of agents m equals 5, 8 and 10, while the ratio r of number of jobs to the number of agents $\left(r=\frac{n}{m}\right)$ is set to 3, 4, 5 and 6 to determine the number of jobs n.
- the cost coefficients, c_{ij} are integers from the uniform distribution U(15,25), while the resource requirements, a_{ij} are integers from U(5,25) and the capacity of agents $b_i=\frac{0.8}{m}\sum_{j\in J}a_{ij}$.

For each job/agent combination, 5 problems are generated, yielding to a total of 60 test problems. These problems have known optimal solutions which are obtained by applying a branch and bound procedure with computation times ranging from a few seconds to several hours [9]. They are used for testing the set partitioning heuristic (SPH) of Cattrysse et al. [10] and the simulated annealing heuristic (FSA) of Cattrysse [8]. These problems can be difficult from computational point of view, see Martello and Toth [42].

For each algorithm, we report the average relative percentage deviation, ARPD, of a heuristic solution, Z_h, from the known optimal solution $Z_{opt}\left\{ARPD=100\times\frac{Z_h-Z_{opt}}{Z_{opt}}\right\}$. We have implemented our algorithms in FORTRAN and report the average CPU time (ACT) and the total CPU time (TCT) in seconds on a VAX-8600 computer.

4.2. λ-neighbourhood descent

We have experimented with five variants of the LS descent methods to investigate the effects of increasing the size of the neighbourhood and different search strategies on the solution quality and the computation requirement. The LS descent methods which start with the Martello and Toth heuristic (MTH) solutions using the λ-generation mechanism with the FA and BA selection strategies are denoted by LS1FA and LS1BA for the case of λ=1, respectively. The LS1FA method followed by another 2-generation descent procedure to produce a 2-opt solution is denoted by LS2FA. The LS1FA and LS1BA implementations with the LT stopping criterion are denoted by LT1FA and LT1BA respectively. The average relative percentage deviation of the best solutions of all runs is denoted by ARPD^b, while the average percentage deviation of all solutions is denoted by ARPD^a. In Tables 1 and 2, the first column shows the twelve sets of test problems. For example, *M05R3* represents a set of 5 test problems, each problem consist of $n=M\times R$ jobs with $M=5$ agents and a given $R=3$. The remaining columns in Table 1 contain the average relative deviations ARPD for the above algorithms while in Table 2 these columns show the algorithms average computation time per set and their total CPU (TCT) times over all runs.

We make a number of observations. Firstly, the LS1FA and LS1BA methods significantly improve the MTH solutions in that the ARPD is reduced from 2.56% to 1.42% by LS1FA and to 1.29% by LS1BA using double the TCT of MTH. This demonstrates the effectiveness of the 1-generation mechanism. Secondly, the LS2FA algorithm requires more ACT than that of the MTH by a factor of ten and more than the LS1FA by factor of 5 with slightly better improved ARPD of 1.33%. The improvement does not justify the extra computation time unless a data structure is used to update the 2-move evaluations. LS2FA produces improvements for the first half of the sets which are of small neighbourhood sizes. As the neighbourhood size in-

Table 1. Average relative percentage deviations for the local search descent methods

Problem set	MTH	LS1FA	LS2FA	LS1BA	LT1FA	LT1BA
M05R3	5.43	2.69	2.32	1.91	1.74	1.61
M05R4	5.02	1.64	1.54	1.13	0.89	0.75
M05R5	2.14	1.58	1.51	1.51	1.26	1.51
M05R6	2.35	0.72	0.66	1.00	0.72	1.01
M08R3	2.63	1.85	1.74	1.63	1.42	1.53
M08R4	1.67	1.05	0.79	1.05	0.82	0.85
M08R5	2.02	1.26	1.11	1.20	1.22	1.18
M08R6	2.45	1.13	1.13	1.11	1.13	1.11
M10R3	2.18	1.73	1.73	1.73	1.48	1.48
M10R4	1.75	1.27	1.25	1.21	1.19	1.13
M10R5	1.78	1.25	1.25	1.15	1.17	0.84
M10R6	1.37	0.88	0.88	0.88	0.81	0.83
ARPD^b	2.56	1.42	1.33	1.29	1.15	1.15
ARPD^a	–	–	–	–	1.38	1.34
NOPT	0	2	2	2	3	3

MTH: Martello and Toth heuristic
LS1FA: MTH+local search with 1-interchange and first admissible
LS2FA: MTH+local search with 2-interchange and first admissible
LS1BA: MTH+local search with 1-interchange and best admissible
LT1FA: Long Term strategy with 1-interchange and first admissible
LT1BA: Long Term strategy with 1-interchange and best admissible
ARPD^a: Average relative percentage deviation of all generated solutions
ARPD^b: Average relative percentage deviation of the best 60 solutions
NOPT: Number of optimal solutions obtained

Table 4. Effects of tabu-list size, FA and BA selection strategies on the performance of tabu search schemes

$\|Ts\|$	$\lceil \frac{n}{6} \rceil$		$\lceil \frac{n}{5} \rceil$		$\lceil \frac{n}{4} \rceil$		$\lceil \frac{n}{3} \rceil$		$\lceil \frac{n}{2} \rceil$	
	TS1	TS6	TS1	TS6	TS1	TS6	TS1	TS6	TS1	TS6
ARPD[c]	0.12	0.21	0.11	0.13	0.09	0.16	0.13	0.21	0.18	0.23
ARPD[b]	0.05	0.06	0.07	0.06	0.03	0.07	0.07	0.09	0.07	0.08
ARPD[a]	0.09	0.13	0.09	0.10	0.07	0.12	0.10	0.15	0.12	0.15
NOPT	40	38	40	40	45	40	36	34	33	32
ACT	14.19	16.30	12.17	15.66	14.23	16.46	12.47	14.01	11.97	14.09
RUNS	118	114	105	116	122	117	115	119	139	137
TCT	1674.4	1858.2	1277.8	1816.6	1736.0	1935.8	1434.0	1667.1	1651.8	1930.3

TS1: Long term; FA selection; R1 tabu restrictions and A1 aspiration criterion
TS6: Long term; BA selection; R1 tabu restrictions and A1 aspiration criterion
Others: as in Table 2
[a–c] as in Table 1

4.4. *Tabu search*

Different implementations can be generated depending on the choices taken for the initial starting solution, the selection strategy, the tabu restrictions, and aspiration criteria. We have implemented six different TS schemes to investigate the effects of these choices on solution quality and CPU time. All the TS stopping criterion with $MAXI=4\times n$ and the same LT stopping criterion described in Sect. 3.3. Six different TS schemes are tested with their choices listed below:

TS1: Long term strategy, FA selection strategy, R1 for tabu restrictions and A1 for aspiration criterion.

TS2: Long term strategy, FA selection strategy, R1 for tabu restrictions and A2 for aspiration criterion.

TS3: TS1 but without the aspiration criterion.

TS4: Long term strategy, FA selection strategy, R2 for tabu restrictions and A1 for aspiration criterion.

TS5: TS1 but without the long term strategy.

TS6: Long term strategy, BA selection strategy, R1 for tabu restrictions and A1 for aspiration criterion.

We have experimented with the above schemes with the aim of identifying the best scheme and highlighting any problems. Firstly, we have investigated the choice between the first admissible (FA) or the classical best admissible (BA) selection strategies. Both FA and BA strategies are embedded into TS1 and TS6 algorithms under the same settings, respectively. we show that there is a good relationship between these strategies and the tabu list size, $|Ts|$. Since the FA strategy updates the tabu-list more frequently than the BA strategy in 1-neighbourhoods with many local solutions, the corresponding TS schemes should not have the same values of $|Ts|$. Experiments are conducted to find out a reasonable range for the tabu list sizes. Computational results are listed in Table 4 for TS1 and TS6 using different values of $|Ts|$, ranging from $\lceil \frac{n}{6} \rceil$ to $\lceil \frac{n}{2} \rceil$, where $\lceil x \rceil$ is the integer rounded value of x.

We observe from the results in Table 4, that for all tabu list values, TS1 yields better quality solutions using less TCT in seconds than TS6. The ARPD[b] of TS1 with $|Ts|=\lceil \frac{n}{4} \rceil$ is 0.03% which is 50% smaller than the ARPD[b] (0.06%) of TS6 with $|Ts|=\lceil \frac{n}{5} \rceil$. Moreover TS1 obtains 45 optimal solutions compared to only 40 by TS6. There is a reasonable value for tabu-list sizes for TS1 and TS6, the quality of solutions deteriorate by moving away from these values in either directions. The average percentage deviation values follow an upward polynomial function with minimum points at $\lceil \frac{n}{4} \rceil$ for TS1 and $\lceil \frac{n}{5} \rceil$ for TS2.

Secondly, we have investigated the dual role of tabu restrictions and aspiration criteria in constraining and guiding the search. Computational results in Table 5 demonstrate that tabu restrictions must be coupled with the right aspiration criteria to serve their purposes. All schemes in Table 5, are variants of the best scheme, TS1, identified in Table 4 with $MAXI=4\times n$ and $|Ts|=\lceil \frac{n}{4} \rceil$. It can be seen that the aspiration criterion would have negative or positive effects on the performance of TS algorithm. For example, replacing the aspiration criterion A1 in TS1 by aspiration criterion A2 in TS2 has caused a huge deterioration in the performance. The ARPD[b] increases from 0.03% for TS1 to 1.09% for TS2 and the number of optimal solutions has dropped from 45 to only 5. TS scheme without aspiration criterion still produce good results provided the right choice of tabu restrictions. For instance, TS3 produces the second best ARPD[b] of 0.05% after TS1. Generally speaking, the choice of tabu restrictions is more important than the choice of aspiration criterion. This is demonstrated by comparing the results of TS1 and TS4. The last point to make on the results in Table 5 is that if the LT strategy can not be afforded computationally, then TS5 without the LT strategy produces good results with an ARPD[b] of 0.08% in one single run on each problem and would be sufficient.

Finally, we have plotted the changes in the average percentage deviations ARPDs of the best, average and worst

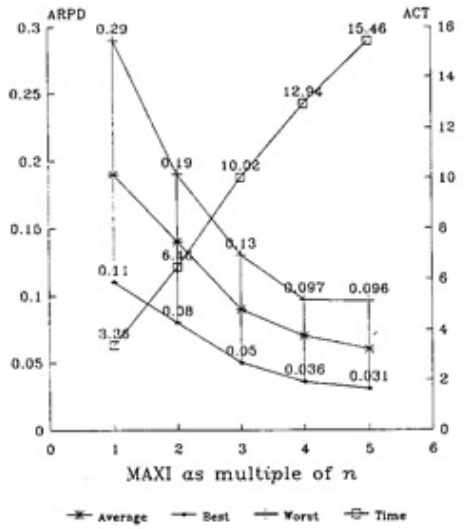

Fig. 3. Effects of TS stopping values, *MAXI*, on the performance of TS1 algorithm

Table 5. Effects of tabu restrictions and aspiration criteria on the performance of TS schemes

	TS1	TS2	TS3	TS4	TS5
ARPD[c]	0.09	1.56	0.12	0.36	0.08
ARPD[b]	0.03	1.09	0.05	0.16	0.08
ARPD[a]	0.07	1.30	0.08	0.26	0.08
NOPT	45	5	39	27	32
ACT	14.23	7.04	13.49	11.17	14.66
RUNS	122	155	103	151	60
TCT	1736.0	1091.20	1389.47	1686.67	879.6

TS1: Long term; tabu restrictions R1 and aspiration criterion A1
TS2: Long term; tabu restrictions R1 and aspiration criterion A2
TS3: Long term; tabu restrictions R1 and no aspiration criterion
TS4: Long term; tabu restrictions R2 and aspiration criterion A1
TS5: TS1 without long term
Others: as in Table 2
[a-c] as in Table 1

solutions with their ACTs in Fig. 3 against increasing values of *MAXI* for the TS stopping criterion. Results with *MAXI* varied from $1\times n$ to $5\times n$ are plotted. From the figure, we draw two conclusions. First, the ARPD of best solutions, ARPD[b], show percentage improvements in a sequence of 27%, 37%, 40% and 13% with percentage increases in the ACT in CPU seconds in a sequence of 92%, 55%, 29% increases in the corresponding computer time and then declines to 13% with 19% increases in the ACT. This indicates that a value of $MAXI=4\times n$ is a good estimate. This is because, beyond $MAXI=4\times n$, there are little improvements to justify the extra increase in the ACT. Second, the error bars represent the differences between ARPD[c] and ARPD[b] at each value of *MAXI*. The difference values level off at values of $MAXI=4\times n$, $5\times n$. We notice that if TS algorithm is allowed to run with a reasonable estimate for its parameters, it converges and produces stable solutions with small gap between the best and worst solutions.

4.5. *Comparison of heuristic*

In this section, we compare the performance of our best implementations of the: Martello and Toth heuristic, MTH; local search descent methods, LT1FA; hybrid simulated annealing/tabu search, RSSA; tabu search schemes TS1 and TS6 against the best algorithms for the GAP in the literature, namely, the simulated annealing FSA of Cattrysse [8], the set partitioning heuristic (SPH) of Cattrysse et al. [10], the branch and bound tree search algorithms of Fisher et al. [16] denoted by FJVBB and of Martello and Toth [42] denoted by MTBB. The results for FJVBB and MTBB are obtained with an upper CPU time limit to terminate the search. The maximum CPU time in seconds on VAX-8600 is set to 250, 750 and 1000 for each problem with $m=5$, 8 and 10 respectively. The results of FSA and SPH are obtained from their references produced on an IBM PS/2 model 80 with 16 MHZ and a 80387 mathematical co-processor with a Microsoft-FORTRAN compiler. A small experiment on bench-mark test problem was conducted to obtain an estimate for the speed factor between VAX and IBM PS/2 computers. It was found that the VAX-8600 is 3.3 faster than the IBM PS/2 model 80. It is however difficult to obtain exact estimate due to many factors (compiler, programming efficiency etc.) but we use the adjusted CPU seconds to draw a reasonable comparison but not to claim absolute accuracy. Computational results of all these algorithms in terms of solution quality are reported in Table 6 while that in terms of computational CPU time requirements are listed in Table 7.

We can observe the following. First, LT1FA improves significantly the solutions of MTH at a reasonably extra CPU time, due to the power of the 1-generation mechanism. LT1FA has the advantage of requiring much smaller CPU time when compared to the more sophisticated methods such as SA or TS algorithms. It could be recommended when the computation time is a limiting resource. Second, RSSA produces better ARPDs (ARPD[b] of 0.04%) than that of SPH, FSA, FJVBB and MTBB. RSSA also requires a considerably smaller amount of CPU time compared to the other algorithms. Further, the numbers of problems not optimally solved by RSSA, SPH, MTBB and FJVBB algorithms are 21, 20, 36 and 34, respectively. Comparing the performance of our simulated annealing RSSA and the Cattrysse's FSA, we see that FSA requires more CPU time than that required by RSSA by a factor of 3.5. FSA produces solutions with ARPD[b] of 0.72% which is much worse than the ARPD[b] (0.04%) of RSSA by a factor of 18. The improved performance of RSSA over FSA might be attributed to the sequential selection of feasible solutions and the non-monotonic schedule as opposed to the random selection and allowance of infeasible solutions in the stepwise cooling schedule used in FSA.

Table 6. Average relative percentage deviation of the presented algorithms

Problem set	MTH	LT1FA	RSSA	TS1	TS6	FSA	SPH	MTBB	FJVBB
*M*05*R*3	5.43	1.74	0.00	0.00	0.00	0.00	0.08	0.00	0.00
*M*05*R*4	5.02	0.89	0.00	0.10	0.24	0.19	0.11	0.00	0.00
*M*05*R*5	2.14	1.26	0.00	0.00	0.03	0.00	0.09	0.00	0.00
*M*05*R*6	2.35	0.72	0.00	0.03	0.03	0.06	0.04	0.18	0.83
*M*08*R*3	2.63	1.42	0.00	0.00	0.04	0.11	0.35	0.00	0.07
*M*08*R*4	1.67	0.82	0.05	0.03	0.00	0.85	0.15	0.52	0.58
*M*08*R*5	2.02	1.22	0.02	0.00	0.02	0.99	0.00	1.32	1.58
*M*08*R*6	2.45	1.13	0.10	0.09	0.14	0.41	0.23	1.32	2.48
*M*10*R*3	2.18	1.48	0.08	0.06	0.06	1.46	0.12	1.06	0.61
*M*10*R*4	1.75	1.19	0.14	0.08	0.15	1.72	0.25	1.15	1.29
*M*10*R*5	1.78	1.17	0.05	0.02	0.02	1.10	0.00	2.01	1.32
*M*10*R*6	1.37	0.81	0.11	0.04	0.07	1.68	0.10	1.55	1.37
ARPD[b]	2.56	1.15	0.04	0.03	0.06	0.72	0.13	0.78	0.84
ARPD[a]	–	1.15	0.21	0.07	0.10	–	–	–	–
ARPD[w]	–	1.38	0.40	0.09	0.13	–	–	–	–
NOPT	0	3	39	45	40	–	40	24	26

ARPD: Average relative percentage deviation of: all generated solution, ARPD[a]; best solutions, ARPD[b] and worst solutions, ARPD[c]
MTH: Martello and Toth [1981] constructive heuristic
LT1FA: Long term descent; 1-interchange mechanism and first admissible
RSSA: Hybrid SA/TS with different seed values
TS1: long term TS; FA selection; R1 tabu restrictions and A1 aspiration criterion
TS6: long term TS; BA selection; R1 tabu restrictions and A1 aspiration criterion
FSA: Cattrysse [1990], fixing simulated annealing algorithm
SPH: Cattrysse et al. [1994], set partitioning heuristic
MTBB: Martello and Toth [1991], branch-and-bound procedure with an upper CPU limit
FJVBB: Fisher et al. [1986], branch-and-bound procedure with an upper CPU limit

Table 7. Average computing time for most of the discussed algorithms

Problem set	RSSA	ANR	TS1	ANR	TS6	ANR	FSA[d]	SPH[d]	MTBB[e]	FJVBB[e]
*M*05*R*3	0.22	1.0	0.73	1.0	0.69	1.0	–	9.79	0.80	11.91
*M*05*R*4	0.91	1.8	1.44	1.0	1.76	1.6	–	21.46	5.96	82.90
*M*05*R*5	1.82	2.0	2.85	1.0	3.68	1.2	–	36.25	52.90	129.63
*M*05*R*6	2.32	1.6	4.86	1.2	5.68	1.8	–	73.75	226.62	250.00
*M*08*R*3	2.48	3.2	3.97	2.0	4.66	2.4	–	30.62	284	260
*M*08*R*4	5.67	3.0	8.85	1.6	8.87	1.6	–	94.37	750	750
*M*08*R*5	13.97	2.6	12.72	2.2	16.25	1.4	–	110.00	750	750
*M*08*R*6	19.53	4.2	22.99	3.6	31.51	2.0	–	363.96	750	750
*M*10*R*3	5.73	4.0	12.19	1.4	10.69	1.6	–	146.67	1000	815
*M*10*R*4	15.55	4.0	18.62	3.2	20.35	2.8	–	373.75	1000	1000
*M*10*R*5	36.96	3.6	34.46	2.2	32.87	2.8	–	721.46	1000	820
*M*10*R*6	65.96	4.4	47.07	3.4	56.53	3.0	–	1931.04	1000	1000
RUNS	177	2.95	122	1.98	166	193	60	60	60	60
ACT	14.26	–	14.23	–	15.66	–	520	326.21	568.32	551.62
Adj. ACT	–	–	–	–	–	–	157.57	98.78	–	–
TCT	2524	–	1736	–	1817	–	9054	5927	34099	33060

[d] Average CPU in seconds on an IBM Model 80 PS/2, 16 Mhz with a 80387 mathematical processor
[e] branch-and-bound procedure with an upper CPU limit of 250, 750, 1000 second for $M = 5$, 8 and 10, respectively
Adj. ACT: Average adjusted CPU in seconds per problem on a VAX-8600 computer
Others: As in previous Tables

Our final task is to compare TS with SA algorithms. The ARPDs of the TS1 algorithm are better than the ARPDs of the RSSA algorithm. However, TS6 is only better than RSSA in terms of ARPD[a] and ARPD[c] whereas RSSA is better in terms of ARPD[b]. Both TS1 and TS6 are faster than RSSA and require a total CPU time of 1736 and 1816 seconds respectively. In general, TS1 produces more stable results due to the small difference of 0.06% between the averages of worst, ARPD[c], and best, ARPD[b]. However, the difference between similar averages in RSSA is relatively big with a value of 0.36%. It should be noted that the quality of the branch and bound heuristics detori-

orate as the problem size increases while the RSSA and TS exhibits a steady behaviour in this respect.

5. Conclusions

We have presented a review of exact and heuristic methods for the GAP. We have shown the importance of the λ-generation mechanism, the LT strategy and the selection strategies in improving the solution quality of the Martello and Toth and the local search methods. We have identified the best design for the hybrid simulated annealing/tabu search (SA/TS) and for the tabu search (TS) heuristics. Effects of various parameters on the performance of the hybrid SA/TS and Tabu search heuristics are investigated. From computational results, it is concluded that the hybrid SA/TS and the TS heuristics are becoming the state-of-the-art heuristics for the GAP and are preferable to other specialized heuristics namely: the set partitioning, simulated annealing and curtailed branch and bound. The superiority of SA and TS is proven both with respect to the solution quality and computational effort on a set of standard test problems from the literature. If the computational time is a limiting factor so that the long term strategy or tabu search can be used, then the local search descent with the 1-generation is recommended.

Certainly more study is warranted, our hybrid SA/TS gives the user control over the trade-off between running time and solution quality through an oscillation behaviour induced by the design of the cooling schedule. The success of this approach suggests the possibility of similarly combining the non-monotonic oscillation approach with other cooling schedules or with other forms of probabilistic guidance (as in probabilistic tabu search, for example) as an alternative to the accept/reject choice rules of simulated annealing and the deterministic memory of most tabu search implementations. The strategic oscillation, we implemented to the objective function control, can be applied similarly to produce enriched search pattern by varying other instrumental parameters, such as numbers of jobs assigned to an agent, quantities of resources used and degrees of infeasibility.

Both SA and TS heuristics consider the entire neighbourhood generated by evaluating $m(m-1)/2$ combination sets of agents. We can always save computational effort, by storing in a special data structure, the best moves in the unchanged sets in order to be used if necessary in the subsequent iteration, without much re-evaluations. This type of data structure is implemented with big savings in computational effort in Osman [48] and Osman and Salhi [52]. Encouraged by the good results obtained by hybrid combinations of genetic algorithms with descent, simulated annealing and tabu search on routing problems (see Thangiah et al. [61] and Thangiah et al. [62]), the author is currently exploring this avenue. Finally, with recent rapid developments in computer technology, we feel that parallel versions of SA and TS would enhance their performance.

Acknowledgement. This research was supported by the Hariri Foundation, Lebanon. The author is grateful to Professor Luk Van Wassenhove for sending me his FJVBB code, Dr. Dirk Cattrysse for sending the standard test problems, the anonymous referees and the editors for constructive comments that helped improving the presentation of this paper.

References

1. Aarts EHL, Korst J (1988) Simulated annealing and boltzmann machines. Wiley and Sons, Chichester
2. Amini MM, Racer M (1994) A rigourous computational comparison of alternative solution methods for the generalized assignment problem. Manag Sci 40:868–890
3. Balas E, Martin CH (1980) Pivot and complement – a heuristic for 0/1 programming. Manag Sci 26:86–96
4. Barcia P, Holm S (1988) A revised bound improvement sequence algorithm. Eur J Oper Res 36:202–206
5. Barcia P, Jörnsten K (1990) Improved Lagrangean decomposition: An application to the generalised assignment problem. Eur J Oper Res 46:84–92
6. Beasley JE (1990) OR-library: distribution test problems by electronic mail. Library email address using anonymous ftp: *mscmga.ms.ic.ac.uk.* J Oper Res Soc 41:1069–1072
7. Benders JF, Van Nunen JA (1983) A property of assignment type mixed linear programming problems. Oper Res Lett 2: 47–52
8. Cattrysse DG (1990) Set partitioning approaches to combinatorial optimization problems. Ph. D. Thesis, Katholieke Universiteit Leuven, Departement Wertuigkunde, Centrum Industrieel Beleid, Belgium
9. Cattrysse DG, Van Wassenhove LN (1992) A survey of algorithms for the generalized assignment problem. Eur J Oper Res 60:260–272
10. Cattrysse DG, Salomon M, Van Wassenhove LN (1994) A set partitioning heuristic for the generalized assignment problem. Eur J Oper Res 72:167–174
11. Collins NE, Eglese RW, Golden BL (1988) Simulated annealing: An annotated bibliography. Am J Math Manag Sci 8: 209–307
12. Dammeyer D, Voß S (1993) Dynamic tabu list management using the reverse elimination method. Ann Oper Res 41:31–46
13. De Werra D, Hertz A (1989) Tabu search techniques: A tutorial and an application to neural networks. OR Spektrum 11: 131–141
14. Dyer M, Frieze A (1990) Probabilistic analysis of the generalized assignment problem. Math Program 55:169–181
15. Fang L, Li T (1990) Design of competition based neural networks for combinatorial optimization. Int J Neural System 3:221–235
16. Fisher M, Jaikumar R, Van Wassenhove L (1986) A multiplier adjustment method for the generalised assignment problem. Manag Sci 32:1095–1103
17. Fisher M, Jaikumar R (1981) A generalised assignment heuristic for the large scale vehicle routing. Networks 11:109–124
18. Gavish B, Pirkul H (1991) Algorithms for the multi-resource generalized assignment problem. Manag Sci 37:695–713
19. Gheysens F, Golden BL, Assad A (1984) A comparison of techniques for solving the fleet size and mix vehicle routing problem. OR Spektrum 6:207–216
20. Glover F (1986) Future path for integer programming and links to artificial intelligence. Comput Oper Res 13:533–549
21. Glover F (1989) Tabu search part I. ORSA J Comput 1:190–206
22. Glover F (1990) Tabu search part II. ORSA J Comput 2:4–32
23. Glover F, Laguna M (1993) Tabu search. In: Modern heuristic techniques for combinatorial problems Reeves CR (ed). Blackwell, Oxford, U.K., pp 70–150
24. Glover F, Laguna M, Taillard E, de Werra D (1993) Tabu search. Annals of Operations Research 41:Baltzer AG, Switzerland
25. Glover F, Taillard E, de Werra D (1993) A user's guide to tabu search. Ann Oper Res 41:3–28

26. Gottlieb ES, Rao MR (1990) (1,K)-Configuration facets for the generalized assignment problem. Math Program 46:53–60
27. Gottlieb ES, Rao MR (1990) The generalized assignment problem – valid inequalities and facets. Math Program 46:31–52
28. Guinard M, Rosenwein MB (1989) An improved dual based algorithm for the generalised assigment problem. Oper Res 17:658–663
29. Hallefjord A, Jörnsten KO, Värbrand P (1993) Solving large scale generalised assignment problem. Oper Res 64:103–104
30. Hasan M, Osman IH (1995) Local search strategies for the maximal planar graph. Int Transact Oper Res 2:(1) forthcoming
31. Jänicke W (1989) Optimal assignment of orders to parallel working subplants without splitting. Computer-Integrated Manufacturing Syst 2:186–187
32. Jörnsten KO, Näsberg M (1986) A new lagrangean relaxation approach to the generalised assignment problem. Eur J Oper Res 27:313–323
33. Jörnsten KO, Värbrand P (1990) Relaxation techniques and valid inequalities applied to the generalized assignment problem. Asia-Pacific Oper Res 7:172–189
34. Jörnsten KO, Värbrand P (1991) A hybrid algorithm for the generalized assignment problem. Optimization 2:273–282
35. Kirkpatrick S, Gelatt JR. CD, Vecchi PM (1983) Optimization by simulated annealing. Science 220:671–680
36. Klastorin TD (1979) An effective sub-gradient algorithm for the generalised assignment problem. Cumput Oper Res 6:155–164
37. Klastorin TD (1979) On the maximal covering location problem and the generalized assignment problem. Manag Sci 25:107–112
38. Laguna M, Kelly JP, Gonzalez-Velarde JL, Glover F (1995) Tabu search for the multilevel assignment problem. Eur J Oper Res 95:(82)170
39. Laporte G, Osman IH (1995) Metaheuristics in combinatorial optimization. Annals of Operations Research, Baltzer AG, Switzerland. (forthcoming)
40. Lee MK (1992) A storage assignment policy in a man-on-board automated storage/retrieval system. Int J Prod Res 30:2281–2292
41. Martello S, Toth P (1981) An algorithm for the generalised assignment problem. In proceedings of the 9th IFORS conference, Hamburg, Germany
42. Martello S, Toth P (1990) Knapsack problems: algorithms and computer implementations. Wiley and Sons, Chichester
43. Martello S, Toth P (1992) Generalized assignment problems. Lecture Notes, In Computer Science 660;351–369
44. Mazzola JB (1989) Generalized assignment with nonlinear capacity interaction. Manag Sci 35:923–941
45. Mazzola JB, Neebe AW (1993) An algorithm for the bottleneck generalized assignment problem. Comput Oper Res 20:355–362
46. Osman IH (1990) A comparison of heuristics for the generalised assignment problem. Research Report, University of Kent, Canterbury. Presented at IFORS 90, Athens
47. Osman IH (1991) Metastrategy simulated annealing and tabu search for combinatorial optimization problems. Ph. D. Thesis, The Management School, Imperial College, University of London
48. Osman IH (1993) Metastrategy simulated annealing and tabu search algorithm for the vehicle routing problem. Ann Oper Res 41:421–451
49. Osman IH, Christofides N (1994) Capacitated clustering problems by hybrid simulated annealing and tabu search. Int Transact Oper Res 1:317–336
50. Osman IH, Laporte G (1995) Modern Heuristics for combinatorial optimization problems. An annotated bibliography. Ann Oper Res (forthcoming)
51. Osman IH, Pott CN (1989) Simulated annealing for permutation flow-shop scheduling. Omega 17:551–557
52. Osman IH, Salhi S (1994) Heuristics for the vehicle fleet mix problem. Proceedings of TRISTAN II, Capri, Italy. Bianco L, (editor) I.A.S.I.-C.N.R. Viale Manzoni 30, 00185 Rome, Italy. Part I:67–72
53. Pesch E, Voß S (1995) Applied local search. OR Spektrum 17:55–65
54. Racer M, Amini M (1994) A robust heuristic for the generalized assignment problem. Ann Oper Res 50:487–503
55. Reeves CR (1993) Modern heuristic techniques for combinatorial problems. Blackwell, Oxford
56. Ross GT, Soland RM (1975) A branch and bound algorithm for the generalised assignment problem. Math Program 8:91–103
57. Ross GT, Soland RM (1977) Modelling facility location problems as generalised assignment problems. Manag Sci 24:345–357
58. Savelsbergh MWP (1993) A branch-and-price algorithm for the generalized assignment problem, Report COC-93-02, Computational Optimization Centre, Georgia Institute of Technology, Atlanta, Georgia, USA
59. Shtub A (198) Modelling group technology cell formation as a generalized assignment problem. Int J Prod Res 27:775–782
60. Skorin-Kapov J (1990) Tabu search applied to the quadratic assignment problem. ORSA J Comput 2:33–45
61. Thangiah SR, Osman IH, Vinayagamoorthy R, Sun T (1993) Algorithms for Vehicle Routing Problems With Time Deadlines. Am J Math Manag Sci 13:323–357
62. Thangiah SR, Osman IH, Sun T (1994) Hybrid Genetic Algorithms, Simulated Annealing and Tabu Search Methods for Vehicle Routing Problems With Time Windows. Working Paper UKC/IMS/OR94/4, Institute of Mathematics and Statistics, University of Kent, Canterbury, UK
63. Trick MA (1992) A linear relaxation heuristic for the generalized assignment problem. Naval Res Logist 39:137–151
64. Van Laarhoven PJM, Aarts EHL (1987) Simulated annealing: theory and applications. Reidel, Doedrecht
65. Zimokha VA, Rubinshtein MI (1988) R & D planning and the generalised assignment problem. Automation and Remote Control 49:484–492