European Journal of Operational Research 207 (2010) 566–578

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A hybrid tabu search/branch & bound approach to solving the generalized assignment problem

Andrew J. Woodcock [a,*], John M. Wilson [b,c]

[a] Nottingham University Business School, NG8 1BB, UK
[b] Loughborough University, LE11 3TU, UK
[c] Aberystwyth University, SY23 2DD, UK

ARTICLE INFO

Article history:
Received 22 August 2008
Accepted 5 May 2010
Available online 10 May 2010

Keywords:
Integer programming
Tabu search
Branch and bound
Generalized assignment problem

ABSTRACT

A new approach for solving the generalized assignment problem (GAP) is proposed that combines the exact branch & bound approach with the heuristic strategy of tabu search (TS) to produce a hybrid algorithm for solving GAP. The algorithm described uses commercial software to solve sub-problems generated by the TS guiding strategy. The TS approach makes use of the concept of referent domain optimisation and introduces novel add/drop strategies. In addition, the linear programming relaxation of GAP that forms part of the branch & bound approach is itself helpful in suggesting which variables might take binary values. Computational results on benchmark test instances are presented and compared with results obtained by the standard branch & bound approach and also several other heuristic approaches from the literature. The results show the new algorithm performs competitively against the alternatives and is able to find some new best solutions for several benchmark instances.

1. Introduction

The generalized assignment problem (GAP) is the problem of either minimizing cost or maximizing profit by assigning n jobs to m agents such that each job is assigned to exactly one agent, whilst ensuring that the resource capacities of each agent are respected. Let $I = \{1,\ldots,m\}$ be the set of agents and $J = \{1,\ldots,n\}$ the set of jobs. GAP can then be formulated as the integer programming problem:

$$\text{GAP} = \text{Minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}, \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij}x_{ij} \leqslant b_i \quad \forall i \in I, \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j \in J, \tag{3}$$

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

where c_{ij} represents the cost of assigning job j to agent i, a_{ij} represents the amount of resource consumed if job j is performed by agent i (in some variants of GAP the amount of resource used by job j is identical for each agent and so a_{ij} is replaced by a_j) and b_i is the amount of available resource for agent i. Knapsack constraints (2) model the resource capacity for each agent $i \in I$, whereas assignment constraints (3) ensure that each job $j \in J$ is assigned to exactly one agent. Constraints (4) impose binary conditions on the variables where x_{ij} takes the value 1 if job j is assigned to agent i, and 0 otherwise. Even instances of GAP with fairly modest values of m and $n$can be quite difficult to solve to optimality using exact algorithms. As values of $m$and n grow further deriving optimal solutions using exact methods becomes unlikely.

GAP is among those combinatorial optimisation problems known to be NP-hard (see Sahni and Gonzalez, 1976 and Fisher et al., 1986) and methods to solve GAP have been extensively researched over the last thirty years due to its relevance to a variety of applications. Some of the less traditional applications include assigning tasks to programmers in software development projects, storage space allocation, designing communication networks, sugar cane harvesting and scheduling television commercials.

Approaches to solving GAP include both exact (branch and bound) and heuristic methods. Earlier work is surveyed in Yagiura and Ibaraki (2004). More recent branch and bound approaches have been proposed by Savelsbergh (1997) who used branch and price to solve GAP, Nauss (2003) who developed bounds and cuts and used variable fixing as part of a branch and bound scheme, and Haddadi and Ouzia (2004), who developed a general branch and bound scheme. On heuristics, many algorithms for s GAP have been presented in the literature including the tabu search (TS) approaches of Laguna et al. (1995) and Diaz and Fernandez (2001) the

* Corresponding author. Tel.: +44 115 8467782.
E-mail addresses: Andrew.woodcock@nottingham.ac.uk, andy@murraystreet.fsnet.co.uk (A.J. Woodcock), j.m.wilson@lboro.ac.uk (J.M. Wilson).

0377-2217/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2010.05.007

Genetic Algorithms (GA) due to Chu and Beasley (1997), Wilson (1997) and Feltl and Raidl (2004), the TS with Simulated Annealing algorithm of Osman (1995), the Path Relinking approaches of Yagiura et al. (2002, 2006), the Variable Depth Search (VDS) approaches of Amini and Racer (1994, 1995) and Yagiura et al. (1998, 1999), a TS method incorporating the use of ejections chains in Yagiura et al. (2004), a variable fixing approach using the linear relaxation of GAP in Trick (1992) and the adaptive search heuristics of Lourenco and Serra (1998).

More recently researchers have used hybrid heuristics or hybrid metaheuristics to solve GAP. Hybrid heuristics combine several heuristics or combine elements of exact methods with heuristics. Examples include Feltl and Raidl (2004), who extended the GA approach of Chu and Beasley (1997) to include using LP-derived starting solutions, and Yagiura and Ibaraki (2004) who survey several hybrid algorithms they developed combining such features as tabu search, ejection chains and path relinking. Further information on recent work on GAP appears in the surveys Nauss (2006) and Yagiura and Ibaraki (2007).

The main purpose of our paper is to show how a new approach for solving GAP, combining a TS heuristic with an exact branch and bound solver, to explore neighbourhoods, can be implemented. The hybrid algorithm was tested on standard sets of benchmark GAP problems and outperformed branch and bound in a commercial solver on medium sized problem instances. On the same set of problems it equalled the best solutions produced by a collection of nine other heuristics from the literature in 10 out of 18 problems. On 27 larger problem instances ($n \geqslant 400$) while the hybrid algorithm did not always perform as well as the best out of the collection of nine heuristics, it equalled the best on four problems but outperformed the best on four problems. Five of the results that were best, or equal best, were in the set of nine problems generally regarded as being some of the most difficult of the 27.

In the next section the motivation behind the hybrid approach to solving GAP will be described, then in Section 3 the phases of the hybrid algorithm, named TSBB, will be described in more detail. In Section 4 computational results are presented and some conclusions are drawn in Section 5.

2. The hybrid approach

Recently researchers have combined aspects of both exact and heuristic approaches to construct more powerful solution methods for solving hard combinatorial optimisation problems. A survey and classification of such approaches is given in Puchinger and Raidl (2005) where the authors classify the approaches into *Integrative* and *Collaborative* methods. The hybrid algorithm presented in Section 2 falls into the integrative category and uses a TS metaheuristic as a master strategy and the branch and bound solver Xpress-MP as a sub-strategy for solving restricted instances of the original problem. Other authors have combined TS with LP or branch and bound. Blue and Bennett (1998) developed a global optimization method using a hybrid of TS and LP descent methods. Bennell and Dowsland (2001) used a hybrid of TS and LP in stock cutting and more recently Lisberg et al. (2009) used LP and TS to solve routing problems. In our approach we will combine TS, LP and branch and bound in a new way to suggest variables to be fixed and unfixed.

2.1. Referent domain optimisation

Referent domain optimisation is proposed by Glover and Laguna (1997). The technique is to fix a large number of the problem variables to produce a restricted integer problem and then to call the branch and bound solver to assign values to the relatively small number of remaining free variables. An application of such an approach is given in Budenbender et al. (2000). In relation to GAP, advantage can be gained from the fact that solving the LP relaxation of GAP yields a solution containing a large number (at least $n - m$) of binary assignments of jobs to agents (Trick, 1992). Tabu memory structures are used to guide the process with regard to the generation of sets of assignments that can be fixed in order to provide restricted regions quickly searchable by the branch and bound solver, and longer term strategies to provide information for intensification and diversification strategies.

2.2. Variable fixing

Several methods are available for variable fixing. Fischetti and Lodi (2003) adopt a *'soft fixing'* approach in order to create subproblems of the original whose solution spaces contain only those solutions within a specified distance of the current solution. This *'soft fixing'* approach is less restrictive and contrasts with the approach of Danna et al. (2005), who intermittently fix the values of certain variables at a particular node of the branch and bound search. Their strategy is to construct and solve a sub-problem every l nodes, where l is large, in an attempt to improve the best solution found during the search.

In contrast to the *'soft fixing'* approach of Fischetti and Lodi (2003) a *'hard-fixing'* approach is incorporated into the hybrid algorithm subsequently described in Section 2. This also takes advantage of the typically large number of binary values assigned to the problem variables in the LP relaxation of GAP, but also considers information contained in previously found integer solutions in order to aid the decision as to which variables to fix when constructing a sub-problem.

2.3. Tabu search

Tabu search, first introduced in a paper by Glover (1986), is a heuristic strategy that has been successfully utilized in order to solve a variety of hard real world optimization problems. In Glover (1989) are the fundamental principles that form the basis of the approach. Glover (1990) presents further refinements and some more sophisticated aspects of the method. Certain basic ingredients need to be incorporated into most TS formulations, and these will be incorporated into TSBB: (a) a suitable initial starting solution, (b) a relevant neighbourhood structure, (c) a suitable tabu list (TL), (d) a suitable evaluation function, and (e) some form of stopping criteria.

3. The TSBB algorithm

The TSBB algorithm takes advantage of a relaxation of GAP, but in contrast to most alternative relaxation approaches relaxes the binary constraints and takes advantage of the fact that the resulting LP relaxation can be easily solved using commercial software (in this case Xpress-MP). Quite importantly the resulting relaxed solution to the problem contains a relatively large number of variables whose values are binary, and thus feasible in relation to the original integer problem. In light of this, it seems intuitive to consider a referent domain optimization approach since the fractional portion of the problem, identified from the relaxed solution, can typically be more easily solved as a smaller sub-problem using the exact branch and bound method. The assignments identified by solving the smaller sub-problem can subsequently be combined with those assignments identified from solving the LP relaxation in order to produce a complete integer feasible solution.

We will first describe the search space and the neighbourhoods used during exploration of this space. We then define both the

short and longer term memory structures used followed by a description of the search process.

3.1. Search space and solution structure

The search space considered for finding a solution to a GAP instance is defined by constraints (2) and (3) containing the set of solutions **x** where each variable x_{ij} represents the assignment of a job j to an agent i with $0 \leqslant x_{ij} \leqslant 1$. A feasible solution to GAP will have $x_{ij} = 0$ or 1 and if $x_{ij} = 1$ then $\sum_{i' \in I} x_{i'j} = 0$ for $i' \neq i$. The neighbourhoods defined in (9) are sub-spaces that are defined in order to strategically search the solution space and are constructed by the addition and relaxation of constraints (4) according to the requirements of the current phase of the search.

3.2. Drop/Add neighbourhood

Given an integer feasible solution **x** to GAP, a move from **x** to $\mathbf{x}'$ can be achieved by first dropping one or more assignments in **x** and then reassigning those jobs that are unassigned as a result of the drop phase. The neighbourhoods used to identify those assignments to be re-allocated to obtain the new solution are termed *Drop/Add* neighbourhoods.

In order to define the *drop* neighbourhood we must first specify the set $S_x = \{(i,j): x_{ij} = 1\}$ determined by **x**. A solution **y** can then be defined to be in the *drop* neighbourhood of **x** if **y** satisfies constraints (6)–(9). Constraint (9) ensures that at least one uniquely assigned job in **x** is no longer uniquely assigned to the same agent in the neighbouring solution **y**, hence at least one assignment has been *dropped*.

$$\sum_{j=1}^{n} a_{ij} y_{ij} \leqslant b_i \quad \forall i \in I, \tag{6}$$

$$\sum_{i=1}^{m} y_{ij} = 1 \quad \forall j \in J, \tag{7}$$

$$0 \leqslant y_{ij} \leqslant 1 \quad \forall i \in I, \quad j \in J, \tag{8}$$

$$\sum_{(i,j) \in S_x} y_{ij} \leqslant |S_x| - 1. \tag{9}$$

The solution $\mathbf{x}'$ can subsequently be defined to be in the neighbourhood of **y** if $\mathbf{x}'$ satisfies constraints (10)–(13) where $S_y = \{(i,j): y_{ij} = 1$ and $(i,j) \in S_x\}$.

$$\sum_{j=1}^{n} a_{ij} x_{ij} \leqslant b_i \quad \forall i \in I, \tag{10}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j \in J, \tag{11}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in I, \quad j \in J, \tag{12}$$

$$x_{ij} = 1 \quad \text{if } (i,j) \in S_y. \tag{13}$$

3.3. Short-term memory

Short-term or *recency-based* memory is used to record historical information about changes in attributes that have taken place within the previous t iterations of the local search and its purpose is to prevent a return to a solution visited within this period and thus prevent the local search from becoming trapped in a cycle. A short-term memory structure might typically take the form of a list containing those attributes forbidden from being included in subsequent solutions. The size of such a list is largely context dependant and takes into account problem size and the strength of the rule that identifies an attribute as being tabu. Experimentation is typically used to determine a value or range of values that are appropriate for setting the size of the tabu list or alternatively allowing the list size to vary dynamically within a suitable range of values as implemented by Wu et al. (2004).

Two short-term memory structures are used in TSBB. The first identifies those assignments dropped as a result of moving from a current solution **x** to an intermediate solution **y**, which are those assignments contained in D where $D = \{(i,j): (i,j) \in S_x$ and $(i,j) \notin S_y\}$. Each entry t^k_{add}, where $k = 1,2,\ldots,t_a$, on the list T_{add}, of size t_a, represents those assignments (i,j) contained in D. Each time the new set D is appended to T_{add} the oldest entry in T_{add} is removed thus maintaining a fixed size tabu list. We define constraint (14) to be a *tabu* constraint and generate a constraint corresponding to each entry on the list T_{add} to be added to sub-problems generated during the search process as subsequently described in 3.5.

$$\sum_{(i,j) \in t^k_{add}} x_{ij} \leqslant |t^k_{add}| - 1. \tag{14}$$

The function of these constraints is to prevent the combination of assignments contained in t^k_{add} being reinstated in the next t_a iterations. The second structure T_{drop} represents those assignments contained in the set $\{(i,j) : (i,j) \in S_{x'}$ and $(i,j) \notin S_x\}$. This tabu restriction prevents those assignments added in the move from **x** to $\mathbf{x}'$ from being dropped in the next iteration. We define constraint (15) to be a *tabu* constraint to be added to sub-problems generated during the search process as subsequently described in 3.5.

$$y_{ij} = 1 \quad \forall (i,j) \in T_{drop}. \tag{15}$$

This tabu restriction is considerably stronger than that preventing assignments being added since there are many more possible assignments excluded from a solution compared to the number of assignments included in a solution. Computational experimentation indicated that T_{drop} should only contain those assignments added during the previous iteration.

3.4. Longer term memory

Longer term memory uses frequency information to introduce *intensification* and *diversification* strategies. These frequency based memory structures are used to provide information to identify promising regions of the solution space previously visited, but which may warrant a more thorough search and identify changes in attributes that would drive the search into unexplored regions of the solution space. The longer term memory structure used by TSBB records how often an assignment is contained in an integer solution. These frequencies are recorded in an $m \times n$ matrix F which is updated at each iteration by increasing the value of the element f_{ij} in F by 1 if an assignment $(i,j) \in S_x$. This structure is used for implementing *intensification* and *diversification* strategies as described in Sections 3.6 and 3.7.

3.5. Searching the neighbourhood

The TSBB algorithm proceeds by first applying the Xpress-MP branch and bound solver to an instance of GAP until a first integer feasible solution is found. The resulting solution is used to initialize $\mathbf{x}^*$, the best solution found during the search, **x** the current and initial solution, and the corresponding objective function values z^* and z respectively. If the branch and bound search completes without finding an integer solution then the problem is declared infeasible and the search ceases. The algorithm proceeds by iteratively performing a *short-term search* phase, followed by an *intensification* phase. If a new improved solution is found during the intensification phase then this solution is used as the starting point for the next short-term phase. If the intensification phase fails to find an improving solution then the next short-term phase begins from a

solution found as a result of implementing a *diversification* phase. The short-term search phase is itself iterative and the stopping condition for the algorithm is simply a time limit that is set dependent upon problem size. The flowchart in Fig. 1 depicts the TSBB algorithm.

The short-term phase begins from a solution **x** and moves to a new solution **x**′ via an intermediate move to a solution **y**. The solution **y** is the optimal solution obtained by solving GAPLR, an LP relaxation of GAP with objective function

$$\text{minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}y_{ij}. \tag{16}$$

Subject to constraints (6), (7), (8), (9) and (15).

Whilst the region defined by (6), (7), (8), (9) and (15) can be extremely large and contain many solutions it is fairly straightforward, in most cases, to obtain an optimal solution to GAPLR using the Xpress-MP solver and hence obtain a good set of assignments S_y. In keeping with a conventional TS approach the short-term phase of the algorithm is allowed to search for non-improving solutions and as a result no cut-off value is specified, and the solution to the relaxed problem is accepted as a partial move from **x** to

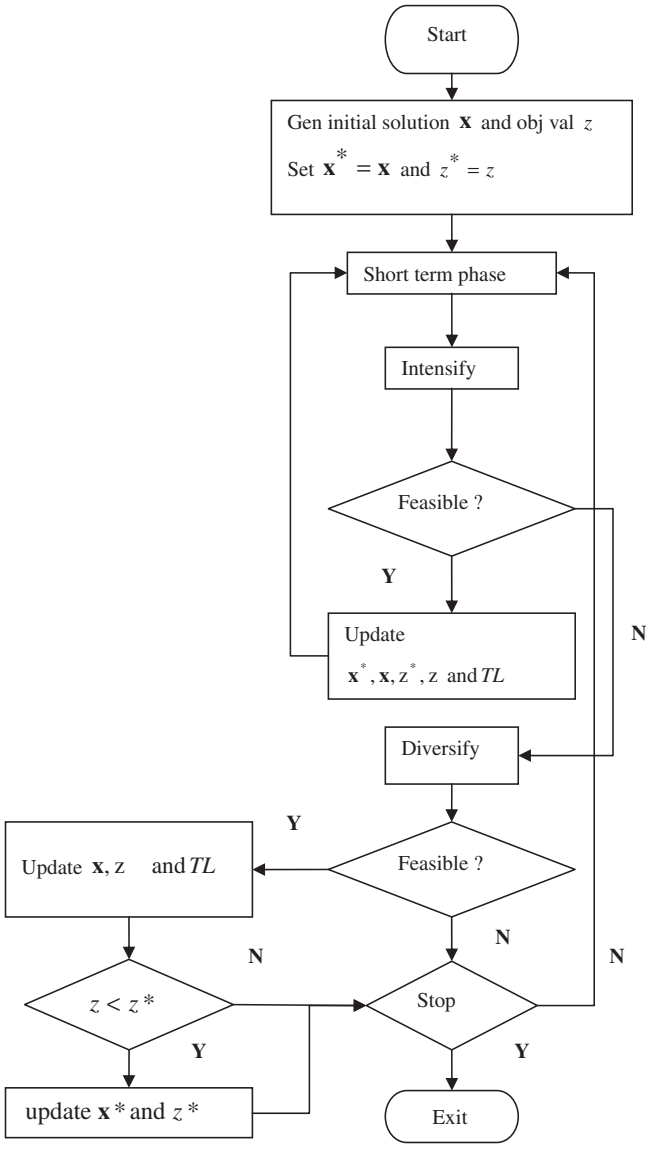

Fig. 1. Flowchart of the TSBB algorithm.

x′ irrespective of the objective function value. If a solution to GAPLR is found then those assignments that have been dropped from **x** by solving GAPLR are added to the tabu list T_{add}. A move from the intermediate solution **y** to the new solution **x**′ is achieved by using **y** to generate a restricted GAP sub-problem RGAP having objective function

$$\text{minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}. \tag{17}$$

subject to constraints (10)–(14).

If GAPLR is infeasible then it is necessary to override the tabu status of one or more assignments that are tabu from being dropped according to the aspiration criteria described in Section 3.8 and update constraint (15) of GAPLR accordingly. If no assignments in **x** are tabu from being dropped then the short-term phase ceases at this point.

Assuming a solution **x**′ has been found by solving RGAP then the tabu list T_{drop} is updated by first deleting those assignments (i,j) currently members of T_{drop} and then recording those assignments present in $S_{x'}$ but not in S_x. The assignments in $S_{x'}$ are the set of assignments added during the current iteration and are therefore tabu from being dropped during the move from **x**′ to the next solution **x**″. In addition to updating T_{drop}, the long-term frequency memory T is also updated at this point. Having obtained a solution **x**′ with corresponding objective function value z', a comparison is made with the objective function value of the best solution z'. If $z' < z^*$, then $\mathbf{x}^*$ and z^* are updated by setting $\mathbf{x}^* = \mathbf{x}'$ and $z^* = z'$ and the new solution is set to be the current solution.

If RGAP is infeasible then clearly the add neighbourhood does not contain a feasible solution to GAP, so no *add* move is made, and instead the solution to the relaxed problem GAPLR is adopted as the new solution **x**′. Accepting the integer infeasible solution (since variables may have fractional values in a feasible solution to GAPLR) to GAPLR is allowed so that the search may continue, and is in keeping with the approach of relaxing problem constraints (see Diaz and Fernandez (2001) and Laguna et al. (1995)) to allow the search to move in and out of the feasible solution space. No manipulation of this fractional solution is performed since the solution typically contains a large number of assignments on which the next formulation of GAPLR is based. A flowchart of the short-term phase is given in Fig. 2. The number of iterations performed during a short-term phase of the algorithm is set according to n/m.

3.6. Intensification

The purpose of the intensification phase is to search a region of the feasible solution space that includes solutions for GAP which contain assignments that could be considered attractive since they are.

(a) Included in the best solution found during the search up to this point, and
(b) Have been included in a large number of integer feasible solutions that have been visited during the search so far.

The best solution $\mathbf{x}^*$ encountered so far by the search is recovered and the assignments (i,j) contained in S_{x^*} are compared with the corresponding values in f_{ij} in order to construct the set $G = \{(i,j): (i,j) \in S_{x^*} \text{ and } f_{ij} \geqslant \alpha r\}$. The parameter α is the threshold for the proportion of time f_{ij}/r, where r is the number of feasible integer solutions generated during the search, that an assignment (i,j) needs to have been included in solutions to date in order to be considered for inclusion in G. At each execution of the intensification phase α is initially set to the value 1 and the set G is subse-

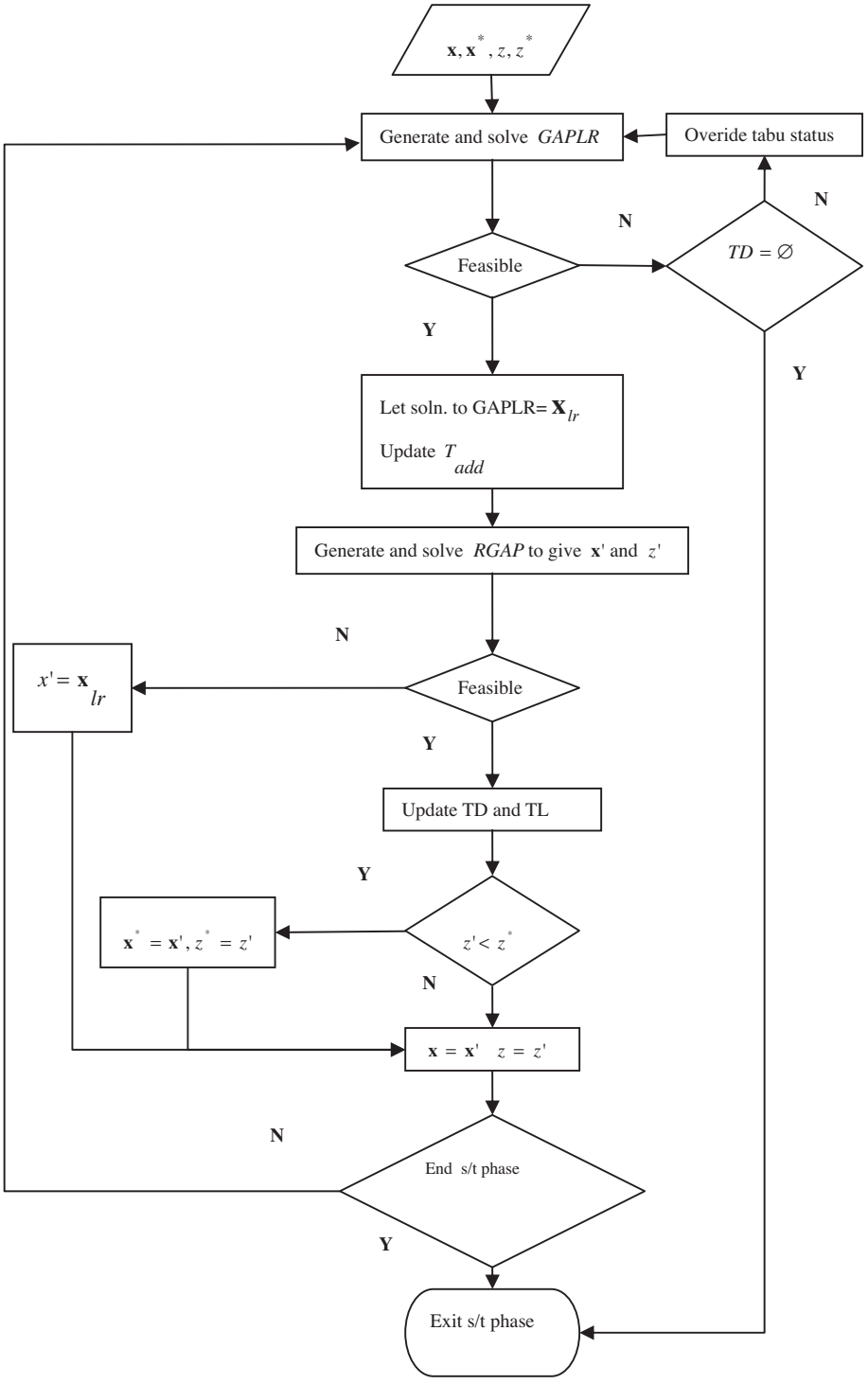

Fig. 2. Short-term phase of TSBB.

quently generated. If G is empty, i.e. no assignments have been included in all feasible integer solutions generated thus far, then α is reduced at a rate of 0.1 at each attempt and another attempt to generate G is carried out. This process continues until G contains at least one assignment or until $\alpha = 0$. In practice the value of α did not reach 0 and the likelihood of such an event occurring is thought to be small, although in some circumstances it may be necessary to re-evaluate the reduction rate of α in order to avoid the occurrence of such an event. The set G can now be used to generate a restricted instance of GAP that has as its feasible region only those solutions that contain all assignments in G by adding an intensifying constraint

$$\sum_{(i,j)\in G} x_{ij} \geqslant |G|, \tag{18}$$

to GAP. (18) is shown as an inequality here for the purposes of presentation within the GAP model, however the constraint is implemented within the algorithm by fixing to 1 the lower bound of

those variables that represent each assignment contained in G. Because good presolve facilities are a standard feature of Xpress-MP, the problem size will be reduced by taking into account the fixed variables and the reduced problem will solve rapidly. Prior to solving the resulting restricted problem a cut-off value of z^* is given to the branch and bound solver which, in keeping with the branch and bound strategy, restricts the search still further. The size of G is determined by α and the size of the feasible region of the restricted problem is determined by the size of G and the quality of the current best solution. Any solution obtained as a result of solving the restricted problem must improve on the best solution found and therefore $\mathbf{x}^*, z^*$ and G are updated accordingly.

3.7. Diversification

TSBB attempts to diversify the search by generating a solution that contains one or more assignments that have very low (ideally 0) residence frequencies, i.e. assignments that have rarely been included in integer feasible solutions found prior to this point in the search. Once such a solution has been generated, it is then used to launch the next short-term phase of the search process. This is achieved by using the longer term frequency based memory to generate a constraint that can be added to GAP that will force the inclusion of low frequency assignments into the diversified solution. To achieve this TSBB first attempts to identify those assignments rarely included in any previous solutions, i.e. those assignments with the lowest (ideally 0) residence frequency values. Then TSBB constructs the set $H = \left\{(i,j) : \frac{f_{ij}}{r} \leqslant h\right\}$, where h is the threshold for the maximum proportion of time that an assignment (i,j) has been included in solutions to date in order to become a member of H, and r is the number of integer solutions found during the search so far. Thus assignments become candidates for inclusion in the diversified solution to be subsequently generated by the addition of the constraint

$$\sum_{(i,j)\in H} x_{ij} \geqslant d, \tag{19}$$

to GAP, where d is the level of required diversification.

If all elements of H have corresponding value $f_{ij} = 0$ then at least one assignment that has never been included in any of the solutions generated will be included in the new diversified solution and hence this new diversified solution will be different to any other solution previously encountered during the search. Clearly the size of H is required to be greater than or equal to d and to achieve the required size for H the threshold h is initially set to 0. If $|H| < d$ then h is increased by 0.1 and H is regenerated. This process continues until $|H| \geqslant d$ at which time the diversification constraint (19) can be added to GAP and the diversified solution generated in the same way as the initial solution described earlier and subsequently used to initiate the next short-term phase of the search. As in the intensification phase the step size for the change in value of the threshold appeared to work quite well in practice although further consideration could be given to this aspect as necessary. If all elements of H have a value $f_{ij} \geqslant 1$ there is no guarantee that the diversified solution will not have been visited previously during the search. Larger values of d will reduce the likelihood of a previously visited solution being generated during the diversification phase since there should be less chance of a larger number of rarely seen assignments being included together in an integer feasible solution. Whilst the quality of the diversified solution is usually poor in comparison to the best known solution, starting the next short-term phase from a previously unseen solution can be advantageous in terms of providing access to previously inaccessible high quality solutions.

3.8. Aspiration criteria

For TSBB, the move from one solution to another within the search is dependent on a feasible solution being found to a relaxed version of GAP, either a feasible solution to the LP relaxation GAPLR when dropping assignments, or the feasible integer solution to an instance of RGAP when attempting to add assignments. In the first of these cases a tabu restriction is applied to force the relaxed solution (which may contain fractional values) to contain those assignments previously added to the current integer solution. When no feasible solution to GAPLR exists, clearly the tabu constraint is too restrictive and so must be relaxed to solve GAPLR. This is achieved by applying an aspiration criterion that replaces constraints (15) with the single constraint (20).

$$\sum_{(i,j)\in T_{drop}} y_{ij} = |T_{drop}| - asp. \tag{20}$$

Initially $asp = 1$ allowing one assignment to be dropped that had previously become tabu from being dropped earlier. As weakening the tabu constraint has the effect of overriding tabu status, it can be classed as applying an aspiration criterion. If the relaxation is still infeasible, asp is repeatedly reduced in value until a feasible solution to it is obtained.

If the solution to an instance of RGAP is found to be infeasible, i.e. no integer solution is generated, then clearly constraint (15) is too restrictive in terms of forcing a collection of assignments to be included in an integer solution and the search must come to a halt. To continue the search the restriction is indirectly overridden by adopting the solution to the relaxation as the current solution, thus forcing one or more assignments in the relaxation to be dropped on the next iteration and thereby relaxing the restriction of constraint (15).

4. Computational results

The algorithm was applied to two sets of benchmark test problems. Benchmark instances for testing GAP have been categorised into five different types A, B, C, D and E and there are a number of instances available at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/ and http://www-or.amp.i.kyoto-u.ac.jp/members/yagiura/gap/.

In terms of difficulty to solve, problems of type A are least difficult and type B and C problems are harder due to tightening of the capacity constraints. Type D and E problems are considerably more difficult due to the inverse relationship between the cost and the capacity coefficients. Problem instances of types A and B can be solved quite easily using Xpress-MP and so do not provide a suitable level of difficulty for testing TSBB. Testing TSBB therefore focuses on problem types C, D and E. The first problem set consists of six problems of each of the three types C, D, E, and are deemed in the literature to be of medium size. These subsets of six have $m = 5$, 10 or 20 and $n = 100$ or 200.

The second set of 27 test problems are deemed to be large problems whose sizes are detailed in Table 3, for which $400 \leqslant n \leqslant 1600$.

All of our testing was carried out on a Viglen CX130 server with a single core Xeon 3.0 GHZ processor and 1.0 GB of RAM, running a Linux operating system. All times are shown in CPU seconds. Details of the speed of the machine used to obtain the results of the other heuristics presented in 4.3 are specified in Yagiura et al. (2004) where the authors suggest that the value SPECint95 taken from the SPEC website (Standard Performance Evaluation Corporation, http://www.spec.org) provides a good indication of the execution speed of their algorithm on different computers with a larger value indicating a faster machine. Whilst no value is

available from the SPECint95 site for the exact specification of our machine, comparison of some machines appearing in both SPECint95 and SPEC2000 indicate approximately a factor of 10 increase from the SPECint95 value to the SPECint2000 value. Although our machine is not specified in SPECint2000 other machines with similar processor specification have values around 1000 to 1500 suggesting a SPECint95 value of between 100 and 150 implying that, by this approximation, our machine is around 10 times faster than that used by Yagiura et al. (2004).

4.1. Parameter settings

The branch and bound results obtained from Xpress-MP (Release 16.01.02) to be presented in 4.2 were all obtained using the default settings of the software from the user manual. These settings were also used by all calls to the Xpress-MP branch and bound solver from within the TSBB algorithm. For each problem instance TSBB takes the parameters t_a and t_d as inputs and all other parameters are set according to problem size, mn.

4.1.1. Tabu tenure

Of the two short-term tabu memory structures used to guide the short-term phase of the algorithm, the first prevents those assignments that have been brought into the current solution as a result of the add phase from being dropped during the next drop phase. Preliminary experimentation indicated that these assignments tend to be highly restrictive, and whilst such a tabu restriction is necessary to avoid cycling there seemed to be no advantage to be gained by using values greater than 1 for this tabu tenure. As a result the computational results detailed in the remainder of this section have all been obtained by fixing the value of t_d to 1. The second short-term memory structure prevents those assignments that are excluded as a result of the previous drop phase from being included during the next t_a add phases. Preliminary experimentation also revealed that very small values of $t_a < 5$ cause obvious cycling of the search in the short-term phase. It was also very clear that increasing values of $t_a > 25$ were causing severe slow down of the search with apparently no benefit to solution quality. This seems fairly intuitive because each of the t_a tabu restrictions takes the form of an additional constraint which tends to slow down the solution of each of the sub-problems in the short-term phase of the algorithm. The preliminary experimentation therefore provided a strong indication that values of t_a in the range $5 \leqslant t_a \leqslant 25$ would be most suitable for testing the TSBB algorithm.

Each problem in the medium size set of test problems has been solved five times with the value of t_a set to 5, 10, 15, 20 and 25. Fig. 3 shows the mean percentage gap from the best bound for the medium sized problem types D and E for each value of t_a. When attempting to solve each problem instance using only Xpress-MP the solver provides a best bound on the problem that can be improved during the run due to the inbuilt strategies of the solver. At the end of each of these solution runs the best bound obtained by Xpress-MP is recorded and used for reporting the following results. The results for problem type C have been omitted since the percentage gap is zero for all values of t_a for this problem set.

Corresponding to Figs. 3, 4 provides the values for large size problems of types C, D and E. These results suggest that the algorithm is fairly robust in terms of solution quality for values of t_a in this range.

The quality of solution for type C problems, however, was found to be consistent across the whole of this range. The two plots also give an indication of the overall level of difficulty for TSBB of the three different problem types. Type D problems seem to provide most difficulty regarding the quality of solution obtainable for medium and large problems.

The TSBB results presented in 4.2 and 4.3 have all been obtained with a single run for each problem instance using a value of $t_a = 15$ (unless otherwise specified) since this value of t_a provided the smallest mean deviation from best bound across all test problems for those values of t_a tested.

4.1.2. Restricted problem size

The set of assignments S_y that are used to create the restricted sub-problems (RGAP) are generated as explained in 3.5 by solving a relaxed problem GAPLR. The average size of the set S_y compared to the number of jobs n was 77.25% for the smaller problem set, 94.4% for the larger problem set and 87.54% across both problem sets with remaining assignments being free to be allocated during the solution run for RGAP.

Due to the size and complexity of many of the benchmark test problems even highly restricted sub-problems can be difficult and time consuming to solve to optimality. It was therefore necessary to restrict the time allowed for the solution of each sub-problem in order to give the algorithm sufficient opportunity to generate the integer feasible solutions in the short-term phase. The short-term phase of TSBB was run for 100 iterations for each of the problems in the medium size test set with a time limit of 60 seconds for each sub-problem. Fig. 5 shows the cumulative frequency of sub-problems that were solved to optimality and obtained an integer feasible solution within a 60 seconds time period, indicating that over 97% of sub-problems were manageable within this time period. The corresponding time limit for larger problem instances was increased proportionally according to the problem size.

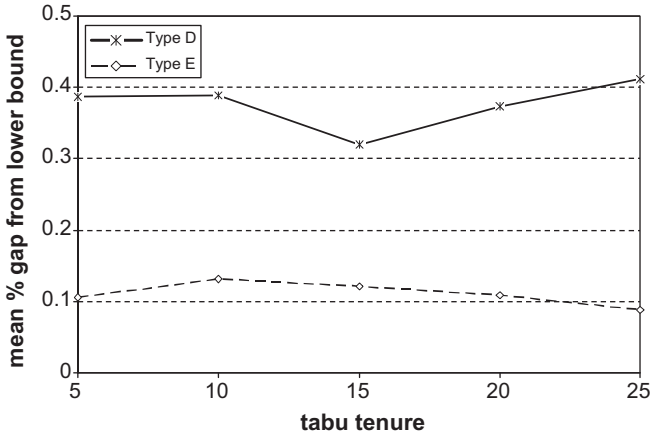

Fig. 3. Mean % gap from best bound for medium size test problems types D and E.

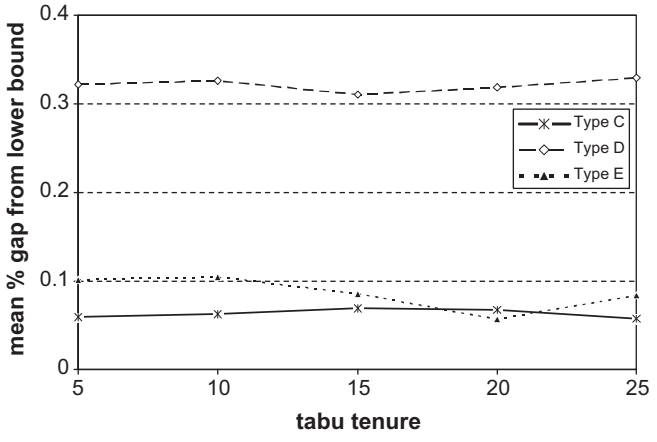

Fig. 4. Mean % gap from best bound for large size test problems.

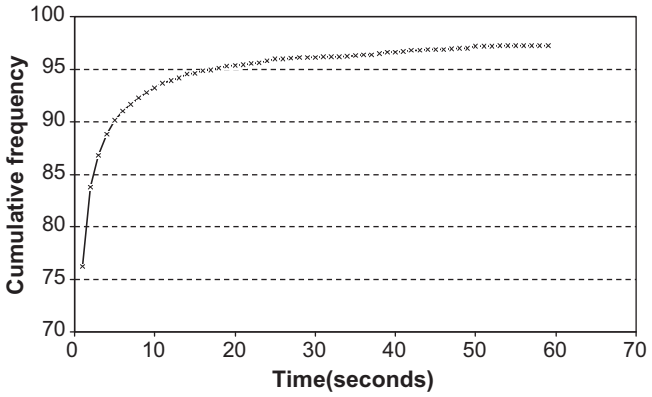

Fig. 5. Cumulative frequency of solution times for short-term phase.

4.2. Comparison with Xpress-MP branch and bound solver

The results presented in this section compare TSBB with the Xpress-MP branch and bound solver. The results obtained by the Xpress-MP solver were all achieved using the default settings of the software for the branch and bound strategy. The same settings were used within TSBB whenever the Xpress solver is called to solve the generated sub-problems.

4.2.1. Medium size problem set

To compare TSBB with Xpress-MP the 18 problems in the medium size problem set were first used. The maximum solution time allowed for each problem with both methods was set at 3000 and 6000 cpu seconds for $n = 100$ and $n = 200$ respectively. These time restrictions were set according to the time allowed for solutions to these problems as defined by Yagiura et al. (2002, 2004). The results in Table 1 show solutions obtained from a single run for each problem by each method with $t_a = 15$ for TSBB.

Xpress was able to solve all of the type C problems and 4 out of the 6 type E problems to proven optimality. The type D problems appear to pose a much stiffer test where only one out of the 6 problems of this type was solved to optimality by Xpress-MP and this was the smallest of the 6. Optimal solutions found by Xpress-MP were also found by TSBB. For the problems not solved to optimality by Xpress-MP, the solution values obtained by TSBB were better than those obtained by Xpress-MP in all instances. The TSBB algorithm tended to find solutions of equally high quality or better, and for some instances in substantially shorter times, suggesting that TSBB is able to outperform the Xpress-MP branch and bound solver for problems of this size and level of difficulty.

Section 3.1 described how each problem was solved by TSBB with 5 different tabu tenures and these results are summarized in Table 2. Although there was some variance in the quality of solution obtained with the different values, the overall quality seemed to be reasonably good across the whole range. The average solution value for each problem across the five different values of the tabu tenure t_a was compared with the solution found by Xpress. For the problems solved to proven optimality by Xpress, TSBB obtained the same optimal solution on each of its five runs. For all other problems in this set the average solution value obtained by TSBB was better than the best solution obtained by Xpress.

4.2.2. Large size problem set

For the large size problems the time limits for each solution run were set at 10,000 cpu seconds for problems where $n = 400$ and $n = 900$ and 50,000 cpu seconds for $n = 1600$. Table 3 compares the best solutions obtained by TSBB and Xpress-MP for the large size problems. The Xpress branch and bound solver was only able to solve 2 out of the 27 problems in this set to proven optimality and both of these optimal solutions were also found by TSBB. In both of these cases TSBB was able to find the optimal solution in a much shorter time than the Xpress solver. Of the remaining 25 problems in this set Xpress-MP was able to find a better objective function value for 4 of these instances, all of which were type D problems. In all of the remaining 21 instances for this problem set TSBB was able to find objective function values at least as good as those found by Xpress.

It is noticeable from results in Table 3 that in some instances the time taken by Xpress-MP to obtain its best solution value is considerably shorter than for TSBB to find its best solution. In all but 2 cases the objective function value of the best solution found by TSBB is smaller than that found by Xpress-MP. If the 14 instances where this occurs are isolated, as presented in Table 4, it can be

Table 1
Comparison of the best solutions obtained by TSBB and Xpress-MP for medium size problems.

Type	m	n	Best Bound*	TSBB			Xpress-MP			
				Best solution value	Time to best solution value	% deviation from best bound	Best solution value	Time to best solution value	% deviation from best bound	Solution optimal? y/n
c	5	100	1931.00	1931	0.79	0.00	1931	0	0.00	y
c	10	100	1402.00	1402	7.25	0.00	1402	6	0.00	y
c	20	100	1243.00	1243	0.69	0.00	1243	1	0.00	y
c	5	200	3456.00	3456	10.36	0.00	3456	4	0.00	y
c	10	200	2806.00	2806	234.62	0.00	2806	233	0.00	y
c	20	200	2391.00	2391	307.57	0.00	2391	871	0.00	y
d	5	100	6351.94	6353	801.3	0.02	6353	1571	0.00	n
d	10	100	6335.03	6350	2581.01	0.24	6359	2453	0.38	n
d	20	100	6160.76	6222	574.74	0.99	6277	1385	1.89	n
d	5	200	12739.87	12,745	271.95	0.04	12,746	4775	0.05	n
d	10	200	12422.19	12,444	383.91	0.18	12,460	122	0.30	n
d	20	200	12224.29	12,280	3400.98	0.46	12,318	5907	0.77	n
e	5	100	12681.00	12,681	18.5	0.00	12,681	84	0.00	y
e	10	100	11577.00	11,577	1026.24	0.00	11,577	327	0.00	y
e	20	100	8423.77	8479	150.4	0.66	8597	3751	2.06	n
e	5	200	24930.00	24,930	58.67	0.00	24,930	20	0.00	y
e	10	200	23307.00	23,307	123.33	0.00	23,307	1156	0.00	y
e	20	200	22376.28	22,393	110.32	0.07	22,658	75	1.26	n
Mean						0.15			0.37	

*Where optimal solution has not been found this bound is the best bound obtained by Xpress during its run.

Table 2
Comparison of average solution for TSBB over 5 runs with best solution obtained by Xpress-MP for medium size problems.

Type	m	n	Best Bound	TSBB			Xpress-MP			
				Average solution value	Average time to best solution value	% deviation from best bound	Best solution value	Time to best solution value	% deviation from best bound	Solution optimal? y/n
c	5	100	1931.00	1931.00	0.80	0.00	1931.00	0.00	0.00	y
c	10	100	1402.00	1402.00	6.99	0.00	1402.00	6.00	0.00	y
c	20	100	1243.00	1243.00	0.70	0.00	1243.00	1.00	0.00	y
c	5	200	3456.00	3456.00	8.63	0.00	3456.00	4.00	0.00	y
c	10	200	2806.00	2806.00	91.81	0.00	2806.00	233.00	0.00	y
c	20	200	2391.00	2391.00	275.79	0.00	2391.00	871.00	0.00	y
d	5	100	6353.00	6353.00	1441.69	0.00	6353.00	1571.00	0.00	n
d	10	100	6335.03	6356.80	940.98	0.34	6359.00	2453.00	0.38	n
d	20	100	6160.76	6230.00	924.50	1.12	6277.00	1385.00	1.89	n
d	5	200	12739.87	12744.00	2499.42	0.03	12746.00	4775.00	0.05	n
d	10	200	12422.19	12445.20	268.49	0.19	12460.00	122.00	0.30	n
d	20	200	12224.29	12292.00	2680.01	0.55	12318.00	5907.00	0.77	n
e	5	100	12681.00	12681.00	175.74	0.00	12681.00	84.00	0.00	y
e	10	100	11577.00	11577.00	351.66	0.00	11577.00	327.00	0.00	y
e	20	100	8423.77	8475.60	689.33	0.62	8597.00	3751.00	2.06	n
e	5	200	24930.00	24930.00	39.88	0.00	24930.00	20.00	0.00	y
e	10	200	23307.00	23307.00	265.80	0.00	23307.00	1156.00	0.00	y
e	20	200	22376.28	22388.00	1584.69	0.05	22658.00	75.00	1.26	n
Mean						0.16			0.37	

Table 3
Comparison of the best solutions obtained for TSBB and Xpress-MP for large size test problems.

Type	m	n	Best Bound*	TSBB			Xpress-MP			
				Best solution value	Time to best solution value	% deviation from best bound	Best solution value	Time to best solution value	% deviation from best bound	Solution optimal? y/n
c	10	400	5597.00	5597	58.62	0.00	5597	255.00	0.00	y
c	20	400	4779.33	4782	2836.58	0.06	4791	8819.00	0.24	n
c	40	400	4243.04	4245	3509.43	0.05	4254	6054.00	0.26	n
c	15	900	11338.38	11,342	5550.57	0.03	11,346	4869.00	0.07	n
c	30	900	9979.12	9993	50.53	0.14	9998	3975.00	0.19	n
c	60	900	9320.43	9337	1876.16	0.18	9459	28.00	1.49	n
c	20	1600	18800.30	18,805	4179.67	0.02	18,811	13.00	0.06	n
c	40	1600	17141.35	17,152	667.99	0.06	17,178	29.00	0.21	n
c	80	1600	16283.00	16,290	23793.17	0.04	16,729	47804.00	2.74	n
d	10	400	24957.38	24,976	656.17	0.07	24,983	12076.00	0.10	n
d	20	400	24556.18	24,631	3746.1864	0.30	24,668	5607.00	0.46	n
d	40	400	24348.00	24,572	5650.05	0.84	24,574	4158.00	0.93	n
d	15	900	55401.16	55,462	2720.93	0.11	55,461	2973.00	0.11	n
d	30	900	54830.23	55,012	16214.35	0.31	54,979	8436.00	0.27	n
d	60	900	54551.00	54,785	3008.51	0.43	54,963	41.00	0.76	n
d	20	1600	97821.97	97,921	49436.21	0.10	97,908	6476.00	0.09	n
d	40	1600	97105.00	97,328	31823.87	0.23	97,311	42646.00	0.21	n
d	80	1600	97034.00	97,449	33495.37	0.43	97,459	24017.00	0.44	n
e	10	400	45746.00	45,746	348.4	0.00	45,746	8293.00	0.00	y
e	20	400	44873.07	44,877	2175.5364	0.01	45,484	3951.00	1.36	n
e	40	400	44548.93	44,640	714.1	0.20	45,484	37.00	2.10	n
e	15	900	102419.27	102,421	1560.29	0.00	102,686	163.00	0.26	n
e	30	900	100423.18	100,500	8099.04	0.08	101,312	18.00	0.89	n
e	60	900	100119.91	100,363	4962	0.24	101,954	6492.00	1.83	n
e	20	1600	180642.64	180,650	3877	0.00	181,060	30095.00	0.23	n
e	40	1600	178286.69	178,394	22614.93	0.06	180,071	17982.00	1.00	n
e	80	1600	176792.84	177,075	3548.02	0.16	178,727	840.00	1.09	n
Mean						0.15			0.64	

*Where optimal solution has not been found this bound is the best bound obtained by Xpress during its run.

seen that in 9 of the 14 instances TSBB is able to find a solution at least as good as that found by Xpress-MP and in shorter times. This behaviour in the Xpress-MP branch and bound approach is typical of how the branch and bound approach can struggle to find improving solutions, particularly for large and difficult problems. In large trees consisting of many thousands of nodes the search process can spend large amounts of time searching areas that subsequently turn out to be unproductive. One of the aims of the TSBB algorithm is to overcome this issue by using TS memory structures and strategies to guide the search towards areas of the solution space that the normal branch and bound process may never reach. The results presented in this section seem to suggest that TSBB

Table 4
Time taken by TSBB to find solutions at least as good as Xpress-MP.

Type	m	n	Best Bound	TSBB		Xpress	
				Obj Value	Time	Obj Value	Time
c	60	900	9320.43	9446	3.86	9459	28.00
c	20	1600	18800.30	18,836	13	18,811	13.00
c	40	1600	17141.35	17,176	23.1	17,178	29.00
d	40	400	24348.00	24,567	2397.87	24,574	4158.00
d	15	900	55401.16	55,461	1332.89	55,461	2973.00
d	30	900	54830.23	55,012	672.00	54,979	8436.00
d	60	900	54551.00	54,986	13.07	54,963	41.00
d	20	1600	97821.97	97,938	4995.67	97,908	6476.00
d	80	1600	97034.00	97,523	33495.4	97,459	24017.00
e	40	400	44548.93	44,876	30.42	45,484	37.00
e	15	900	102419.27	102,571	13.2	102,686	163.00
e	30	900	100423.18	100,922	16.83	101,312	18.00
e	40	1600	178286.69	178,406	5674.12	180,071	17982.00
e	80	1600	176792.84	177,445	736.32	178,727	840.00

achieves reasonable success in achieving this aim since it is able to find new improving solutions in later search stages where branch and bound cannot.

As with the medium size set of problems the large problems were each run a total of 5 times with the different values of the tabu tenure t_a . The average solution value for each problem over the 5 runs is presented in Table 5, in order to compare these values with the best solutions obtained by the Xpress-MP branch and bound solver. The average time taken to obtain the best solution in each case is given in column 6 of Table 5. The average solution values obtained across the five different values of t_a compare favourably with the best solutions found by Xpress, reinforcing the fact that the TSBB algorithm performs well compared to Xpress branch and bound across this particular range of values for t_a.

4.3. Comparison with heuristic methods

This section provides a comparison of the TSBB algorithm with several heuristic approaches that have been applied to the two problem sets for GAP. The first of these comparisons in Section 4.3.1 compares TSBB with 9 alternative algorithms that have all been tested on the medium size problem set while 4.3.2 compares TSBB with 4 other approaches whose performance has been tested on the large problem set.

4.3.1. Comparison of alternative algorithms for medium size problems

The TSBB algorithm is compared with 9 other heuristics for the medium size problems and the solution values for each of these is given in Table 6 where values in bold type indicate the best solu-

Table 5
Comparison of TSBB over 5 runs with best solution obtained by Xpress-MP for large size problems.

Type	m	n	Best Bound	TSBB			Xpress-MP			
				Average solution value	Ave time to best solution	% dev from best bound	Best solution value	Time to best solution	% dev from best bound	Solution optimal? y/n
c	10	400	5597.00	5597.00	281.14	0.00	5597	255.00	0.00	y
c	20	400	4779.33	4782.20	1336.68	0.06	4791	8819.00	0.24	n
c	40	400	4243.04	4245.00	1010.81	0.05	4254	6054.00	0.26	n
c	15	900	11338.38	11341.40	3444.62	0.03	11,346	4869.00	0.07	n
c	30	900	9979.12	9991.40	1657.99	0.12	9998	3975.00	0.19	n
c	60	900	9320.43	9333.60	1763.68	0.14	9459	28.00	1.49	n
c	20	1600	18800.30	18807.00	4932.39	0.04	18,811	13.00	0.06	n
c	40	1600	17141.35	17151.80	1091.61	0.06	17,178	29.00	0.21	n
c	80	1600	16283.00	16290.8	26748.59	0.05	16,729	47804.00	2.74	n
d	10	400	24957.38	24977.60	2346.78	0.08	24,983	12076.00	0.10	n
d	20	400	24556.18	24630.20	5469.28	0.30	24,668	5607.00	0.46	n
d	40	400	24348.00	24563.60	5040.72	0.89	24,574	4158.00	0.93	n
d	15	900	55401.16	55460.20	5831.54	0.11	55,461	2973.00	0.11	n
d	30	900	54830.23	55008.6	6671.97	0.33	54,979	8436.00	0.27	n
d	60	900	54551.00	54833.40	2914.67	0.52	54,963	41.00	0.76	n
d	20	1600	97821.97	97918.20	36113.54	0.10	97,908	6476.00	0.09	n
d	40	1600	97105.00	97355.40	27647.99	0.26	97,311	42646.00	0.21	n
d	80	1600	97034.00	97500.20	16064.41	0.48	97,459	24017.00	0.44	n
e	10	400	45746.00	45746.00	426.09	0.00	45,746	8293.00	0.00	y
e	20	400	44873.07	44893.40	1665.03	0.05	45,484	3951.00	1.36	n
e	40	400	44548.93	44633.60	4582.97	0.19	45,484	37.00	2.10	n
e	15	900	102419.27	102421.00	2394.00	0.00	102,686	163.00	0.26	n
e	30	900	100423.18	100504.40	6544.70	0.08	101,312	18.00	0.89	n
e	60	900	100119.91	100375.40	7644.392	0.26	101,954	6492.00	1.83	n
e	20	1600	180642.64	180646.40	7729.54	0.00	181,060	30095.00	0.23	n
e	40	1600	178286.69	178408.6	20299.16	0.07	180,071	17982.00	1.00	n
e	80	1600	176792.84	177058.80	28657.86	0.15	178,727	840.00	1.09	n
Mean						0.16			0.64	

Table 6
Solution values of the different heuristics for the medium size problems.

Type	m	n	Best Bound	TSBB (s)	TSBB (L)	PREC	ECTS	BVDS-i	BVDS-j	VDS	RA	LKGG	CB	DF
C	5	100	1931	**1931**	**1931**	**1931**	**1931**	**1931**	**1931**	**1931**	**1938**	**1931**	**1931**	**1931**
	10	100	1402	**1402**	**1402**	**1402**	**1402**	**1402**	1403	**1402**	1405	1403	1403	**1402**
	20	100	1243	**1243**	**1243**	**1243**	**1243**	1244	1244	1246	1250	1245	1244	**1243**
	5	200	3456	**3456**	**3456**	**3456**	**3456**	**3456**	3457	3457	3469	3457	3458	3457
	10	200	2806	**2806**	**2806**	2807	**2806**	2809	2808	2809	2835	2812	2814	2807
	20	200	2391	2393	**2391**	**2391**	2392	2401	2400	2405	2419	2396	2397	**2391**
D	5	100	6349	6356	6356	**6353**	6357	6358	6362	6365	n.a	6386	6373	6357
	10	100	6335.03	6374	6366	6356	6358	6367	6370	6380	6532	6406	6379	**6355**
	20	100	6160.76	6281	6254	**6211**	6221	6275	6245	6284	6428	6297	6269	6220
	5	200	12,742	12,756	12,745	**12,744**	12,746	12,755	12,755	12,778	n.a	12,788	12,796	12,747
	10	200	12422.19	12,459	12,456	**12,438**	12,446	12,480	12,473	12,496	12,799	12,537	12,601	12,457
	20	200	12224.29	12,351	12,332	**12,269**	12,284	12,440	12,318	12,335	12,665	12,436	12,452	12,351
E	5	100	12,681	**12,681**	**12,681**	**12,681**	12,682	**12,681**	12,682	12,685	12,917	12,687	n.a	**12,681**
	10	100	11,577	11,597	11,584	**11,577**	**11,577**	11,585	11,599	11,585	12,047	11,641	n.a	11,581
	20	100	8423.77	8528	8479	8444	**8443**	8499	8484	8490	9004	8522	n.a	8460
	5	200	24,930	**24,930**	**24,930**	**24,930**	**24,930**	24,942	24,933	24,948	25,649	25,147	n.a	24,931
	10	200	23,307	23,309	**23,307**	23,310	**23,307**	23,346	23,348	23,340	24,717	23,567	n.a	23,318
	20	200	22376.28	22,403	22,393	**22,379**	22,391	22,475	22,437	22,452	24,117	22,659	n.a	22,422

tion found out of all the methods. The values for the 9 heuristics are taken from results reported in Table 2 of Yagiura et al. (2006) for a single run for each problem instance, where the problems with $n = 100$ were limited to 150 seconds of solution time and the problems with $n = 200$ were limited to 300 seconds (on a Sun Ultra 2 Model 2300), except for VDS on a few type D and E instances, LKGG on all type E instances, and CB and DF on all instances. Table 6 gives two sets of solution values for TSBB, column 5 headed TSBB (s) and column 6 headed TSBB (L). The solution values under the TSBB (s) column were obtained with a time limit of 15 seconds for problems with $n = 100$ and 30 seconds for problems with $n = 200$ (to allow for direct comparison considering difference in computer speed i.e. factor of 10 difference), while the values in the column headed TSBB (L) were obtained with time limits identical to those specified in Yagiura et al. (2006) i.e. 150 seconds for $n = 100$ and 300 seconds for $n = 200$. The values for TSBB were obtained from a single run for each instance with $t_a = 15$. The other heuristic methods are:

- Path relinking with ejection chains (PREC) by Yagiura et al. (2006).
- Ejection chain tabu search (ECTS) by Yagiura et al. (2004).
- Two branching variable depth search methods from Yagiura et al. (1998).
- A variable depth search method (VDS) due to Yagiura et al. (1998)
- A variable depth search approach (RA) by Amini and Racer (1994).
- A TS approach (LKGG) of Laguna et al. (1995) reported in Yagiura et al. (2004).
- A genetic algorithm approach (CB) due to Beasley and Chu (1997).
- The tabu search method (DF) by Diaz and Fernandez (2001).

The summary of results in Table 6 show that the results obtained by TSBB compare very favourably against all of the other heuristics for both of the imposed time limits. The results highlight that increasing the allowable solution time produces very small benefit in terms of solution quality, 0.014% on average for the type C problems, 0.14% for the type D problems, 0.12% for the type E problems and 0.09% overall types.

4.3.2. Comparison of alternative algorithms for large size problems

Fewer methods in the literature have been tested on the set of large test problems and so the comparison is restricted to 5 alternative algorithms. These are the ejection chain TS, path relinking with ejection chain, branching variable depth search, the tabu search LKGG and the Diaz and Fernandez TS. Table 7 gives solution values for each of the 5 algorithms. The results for TSBB are reported for a short-time limit in the column headed TSBB (s) (1000 seconds for $n \leqslant 900$ and 5000 for $n = 1600$) and a longer time limit in the column headed TSBB (L) (10,000 seconds for $n \leqslant 1600$ and 50,000 seconds for $n = 1600$) and have been obtained from a single run for each problem instance with $t_a = 15$. The solution values for the other 5 algorithms are those reported in Table 3 of Yagiura et al. (2006) for a single run for each problem instance (on a Sun Ultra 2 Model 2300). As shown in Table 7, TSBB performs moderately well on the type C problems, less well on the type D problems, but particularly strongly on the difficult type E problems, where our algorithm outperforms all the alternative methods on 2 problem instances. A factor of 10 increase in solution time however typically results in small average improvements in solution quality, 0.015% for type C problems, 0.091% for type D problems, 0.031% for type E problems and 0.046% overall. The results obtained during the longer solution runs were improved, compared to the results of the shorter runs for 5 type C problems, 8 type D and 5 type E problems.

Fig. 6 illustrates how the solution is improved over time for a large problem of type C with $m = 80$ and $n = 1600$ where we see substantial initial improvement within the first 100 seconds and smaller improvements after longer computational periods.

4.3.3. Summary

The results presented here show that a general branch and bound solver struggles to provide optimal solutions to benchmark GAP instances particularly as problem size and difficulty increases where the quality of solutions can be poor. Combining the TS strategy with the branch and bound solver however produces significant improvements in terms of solution quality and the length of time taken to obtain these solutions. Comparisons between TSBB and other heuristic approaches from the literature indicates that the proposed TSBB algorithm performs well for the set of benchmark problem instances although noticeably algorithm PREC seems to perform particularly well on the type D problems due to the importance it places on the diversification aspect of the search. In Yagiura et al. (2006) experimental results indicate that the distance between locally optimal solutions for the type D problems are further apart than those for types C and E problems and so the strong diversification element of PREC attempts to overcome this.

Table 7
Solution values of the different heuristics for the large size problems.

Type	*m*	*n*	Best Bound	TSBB (s)	TSBB (L)	ECTS	PREC	BVDS	LKGG	DF
C	10	400	5597	**5597**	**5597**	**5597**	**5597**	5605	5608	5598
	20	400	4779.33	4783	**4782**	**4782**	**4782**	4795	4792	4786
	40	400	4243.04	4246	**4245**	**4244**	**4245**	4259	4251	4248
	15	900	11338.38	11,345	11,342	**11,341**	**11,341**	11,368	11,362	n.a
	30	900	9979.12	9993	9993	9985	**9984**	10,022	10,007	n.a
	60	900	9320.43	9339	9337	**9328**	**9328**	9386	9341	n.a
	20	1600	18800.3	18,805	18,805	**18,803**	**18,803**	18,892	18,831	n.a
	40	1600	17141.35	17,152	17,152	17,147	**17,145**	17,262	17,170	n.a
	80	1600	16,283	16,297	16,290	16,291	**16,289**	16,380	16,303	n.a
D	10	400	24957.38	24,978	24,976	24,974	**24,969**	25,032	25,145	25,039
	20	400	24556.18	24,665	24,631	24,614	**24,587**	24,780	24,872	24,747
	40	400	24,348	24,589	24,572	24,463	**24,417**	24,724	24,726	24,707
	15	900	55401.16	55,465	55,462	55,435	**55,414**	55,614	56,423	n.a
	30	900	54830.23	55,012	55,012	54,910	**54,868**	55,210	55,918	n.a
	60	900	54,551	54,986	54,785	54,666	**54,606**	55,123	55,379	n.a
	20	1600	97821.97	97,938	97,921	97,870	**97,837**	98,248	100,171	n.a
	40	1600	97,105	97,467	97,328	97,177	**97,113**	97,721	99,290	n.a
	80	1600	97,034	97,523	97,449	97,109	**97,052**	98,146	98,439	n.a
E	10	400	45,746	**45,746**	**45,746**	**45,746**	**45,746**	45,878	172,185	45,781
	20	400	44873.07	44,899	**44,877**	44,882	44,879	45,079	137,153	45,007
	40	400	44548.93	44,640	44,640	44,589	**44,574**	44,898	63,669	44,921
	15	900	102419.3	102,428	**102,421**	102,423	102,422	102,755	463,142	n.a
	30	900	100423.2	100,591	100,500	100,442	100,434	100,956	527,451	n.a
	60	900	100119.9	100,441	100,363	100,185	**100,169**	100,917	479,650	n.a
	20	1600	180642.6	180,650	180,650	180,647	180,646	181,143	936,609	n.a
	40	1600	178286.7	178,500	178,394	178,311	**178,302**	179,036	1,026,259	n.a
	80	1600	176792.8	177,075	177,075	176,866	**176,857**	178,205	1,026,417	n.a

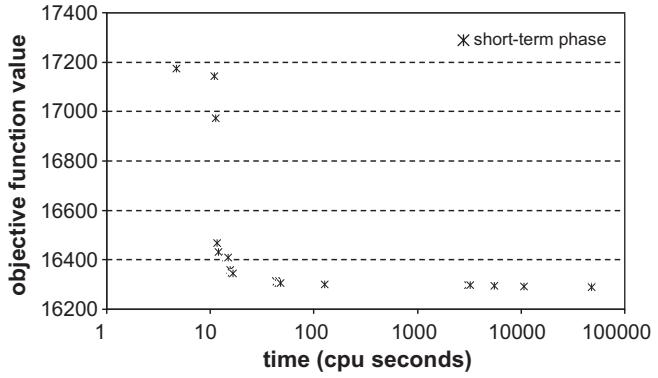

Fig. 6. New best solutions found by search phase for problem type C, m = 80, n = 1600.

5. Conclusions

A new hybrid algorithm for solving GAP has been presented and its performance compared with alternative algorithms from the literature.

5.1. TSBB development and performance

Because the TSBB approach utilises a commercial mathematical programming software solver, Xpress-MP, this aids implementation of the standard branch and bound aspect of the algorithm. This has been important since the TSBB algorithm has been coded in C thus allowing the generated sub-problems to be easily called from within the program at the relevant stages of the algorithm. The algorithm remains easy to implement and is able to take advantage of state of the art branch and bound facilities within Xpress-MP.

In Section 3.2 detailed results of a comparison of TSBB with the Xpress-MP solver on two sets of benchmark test problems are presented. The objective function values of the best obtainable solutions by TSBB generally tend to be lower than those found by Xpress. By focusing the search based on information gained from previous integer feasible solutions the neighbourhoods defined by the subsequent sub-problems tend to be more fruitful than the areas of the branch and bound tree that are chosen to be searched by the default settings of the Xpress software. In all the instances where Xpress is able to obtain an optimal solution to a problem, TSBB is also able to find the optimal solution, usually in significantly shorter periods of time than Xpress. These results indicate that the memory structures utilised within the different phases of the TSBB algorithm are effective at exploiting the solution space and defining sub-trees that provide high quality solutions.

The comparisons in Section 3.3 also give a good indication that TSBB is competitive against a range of different approaches reported in the literature in recent years. In contrast to those of 4.2 a comparison with alternative algorithms with a much shorter time limit is performed. The results of these comparisons show that TSBB is competitive against all the other algorithms and can therefore be considered as a valid approach to solve large hard instances of GAP over both longer and shorter time periods.

The TSBB algorithm has practical relevance since it has been shown to perform well on benchmark instances of varying size and difficulty and there is evidence to show that it is competitive with regard to the quality of solutions obtained. TSBB also has theoretical relevance since it shows how it is possible to use TS memory structures to guide a standard branch and bound process by defining neighbourhoods that are represented by sub-problems that can be passed to an existing commercial solver to be solved, and that using the default settings can produce high quality solutions.

5.2. Further work

There are two aspects to the further development of the TSBB algorithm. The first is the development of the algorithm itself and second the suitability and application of TSBB to extensions to the GAP and other problems.

Some large and tightly constrained sub-problems are extremely challenging to solve and so additional strategies at the sub-problem level may aid the solution process. Such strategies may involve fixing additional variables and creating additional tabu structures using a layered approach (see Glover, 1990) at the sub-problem level. An alternative form of memory that may be of interest might be to record the frequency of assignments appearing in an elite solution list containing the best n solutions say. This information could then be used as a means of implementing an alternative intensification strategy, for example.

TSBB has used the default settings of Xpress-MP for the branch and bound aspect of the algorithm and it may be fruitful to adjust some of these settings either dynamically within the search process or statically prior to attempting to solve problems of different types and sizes and perhaps the structure of the problem could be further exploited in doing so.

As well as adapting TSBB by means of differing strategies with regard to its implementation a further aspect that may be worthy of investigation would be to attempt to apply TSBB to extensions of the GAP problem, including multilevel GAP problems (Laguna et al., 1995) and GAPs with special ordered sets (de Farias et al., 2000).

References

Amini, M.M., Racer, M., 1994. A rigorous computational comparison of alternative solution methods for the generalised assignment problem. Management Science 40, 868–890.

Amini, M.M., Racer, M., 1995. A hybrid heuristic for the generalized assignment problem. European Journal of Operational Research 87, 343–348.

Bennell, J., Dowsland, K.A., 2001. Hybridising tabu search with optimization techniques for irregular stock cutting. Management Science 47, 1160–1172.

Blue, J.A., Bennett, K.P., 1998. Hybrid extreme point tabu search. European Journal of Operational Research 106, 676–688.

Budenbender, K., Grunert, T., Sebastian, H., 2000. A Hybrid Tabu Search/Branch-and-Bound Algorithm for the Direct Flight Network Design Problem. Transportation Science 34 (4), 364–380.

Chu, P.C., Beasley, J.E., 1997. A Genetic Algorithm for the generalised assignment problem. Computers and Operations Research 27, 17–23.

Danna, E., Rothberg, E., Le Pape, C., 2005. Exploring relaxation induced neighbourhoods to improve MIP solutions. Mathematical Programming A (102), 71–90.

de Farias Jr., I.R., Johnson, E.L., Nemhauser, G.L., 2000. A generalized assignment problem with special ordered sets: A polyhedral approach. Mathematical Programming A (89), 187–203.

Diaz, J.A., Fernandez, E., 2001. A Tabu search heuristic for the generalized assignment problem. European Journal of Operational Research 132, 22–38.

Feltl, H., Raidl, G.R., 2004. An improved hybrid genetic algorithm for the generalized assignment problem. In: Haddad, H.M. et al. (Eds.), Proceedings of the 2003 ACM Symposium on Applied Computing. ACM Press, pp. 990–995.

Fischetti, M., Lodi, A., 2003. Local branching. Mathematical Programming B (98), 23–47.

Fisher, M.L., Jaikumar, R., Van Wassenhove, L.N., 1986. A multiplier adjustment method for the generalized assignment problem. Management Science 32, 1095–1103.

Haddadi, S., Ouzia, H., 2004. Effective algorithm and heuristic for the generalized assignment problem. European Journal of Operational Research 153, 184–190.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research 13, 33–549.

Glover, F., 1989. Tabu search-part I. ORSA Journal on Computing 1 (3), 190–205.

Glover, F., 1990. Tabu search-part II. ORSA Journal on Computing 2 (1), 4–32.

Glover, F., Laguna, M., 1997. Tabu Search. Kluwer Academic Publishers, Boston.

Laguna, M., Kelly, J.P., Gonzalez-Verlade, J.L., Glover, F., 1995. Tabu search for the multilevel generalized assignment problem. European Journal of Operational Research 82, 176–189.

Lisberg, P., Liden, B., Ronnqvist, M., 2009. A hybrid method based on linear programming and tabu search for routing logging trucks. Computers and Operations Research 36, 1122–1144.

Lourenco, H.R., Serra, D., 1998. Adaptive approach to heuristics for the generalized assignment problem. Technical Report. Department of Economics and Management, Universitat Pompeu Fabra, Roman Trias Fargas 25-27, 08005 Barcelona, Spana. <http://www.econ.upf.es/dechome/what/wpapers/listwork.html>.

Nauss, R.M., 2003. Solving the generalised assignment problem: An optimizing and heuristic approach. INFORMS Journal on Computing 15 (3), 249–266.

Nauss, R.M., 2006. The generalized assignment problem. In: Karloff, J.K. (Ed.), Integer Programming Theory and Practice. CRC Press, pp. 39–55.

Osman, I.H., 1995. Heuristics for the generalised assignment problem: Simulated annealing and tabu search approaches. OR Spektrum 17, 211–215.

Puchinger, G., Raidl, G., 2005. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation. Springer. pp. 41–53.

Sahni, S., Gonzalez, T., 1976. *P*-complete approximation problems. Journal of ACM 23, 555–565.

Savelsbergh, M., 1997. A branch-and-price algorithm for the generalized assignment problem. Operations Research 45, 831–841.

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

Wilson, J.M., 1997. A genetic algorithm for the generalised assignment problem. Journal of the Operational Research Society 48, 804–809.

Wu, T.H., Yeh, J.Y., Syau, Y.R., 2004. A tabu search approach to the generalised assignment problem. Journal of the Chinese Institute of Industrial Engineers. 21 (3), 301–311.

Xpress-MP, Fair Isaac Ltd., Blisworth, Northamptonshire, United Kingdom.

Yagiura, M., Ibaraki, T., 2004. Recent metaheuristics for the generalized assignment problem. Proceedings of the 12th International Conference on Informatics Research for Development of Knowledge Society Infrastructure (IKS'04). IEEE.

Yagiura, M., Ibaraki, T., 2007. Generalized assignment problem. In: Gonzalez, T.F. (Ed.), Handbook of Approximation Problems and Metaheuristics. Chapman and Hall/CRC. Chapter 48.

Yagiura, M., Yamaguchi, T., Ibaraki, T., 1998. A variable depth search algorithm with branching search for the generalised assignment problem. Optimisation Methods & Software 10, 419–441.

Yagiura, M., Yamaguchi, T., Ibaraki, T., 1999. A variable depth search algorithm for the generalized assignment problem. In: Voss, S., Martello, S., Osman, I.H., Roucairol, C. (Eds.), Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization. Kluwer Academic Publishers, Boston, pp. 459–471.

Yagiura, M., Ibaraki, T., Glover, F., A path relinking approach for the generalized assignment problem. In: Proceedings of the International Symposium on Scheduling, Japan, June 4–6, 2002, pp. 105–108.

Yagiura, M., Ibaraki, T., Glover, F., 2004. An ejection chain approach for the generalized assignment problem. INFORMS Journal on Computing 16 (2), 131–151.

Yagiura, M., Ibaraki, T., Glover, F., 2006. A path relinking approach with ejections chains for the generalized assignment problem. European Journal of Operational Research 169 (2), 548–569.