PERGAMON

Computers & Operations Research 28 (2001) 1039–1048

computers & operations research

www.elsevier.com/locate/dsw

A dynamic tabu search for large-scale generalised assignment problems

A.J. Higgins*

CRC for Sustainable Sugar Production, CSIRO Tropical Agriculture, 306 Carmody Road, St. Lucia 4067, Australia

Received 1 May 1999; received in revised form 1 October 1999

Abstract

A new tabu search (TS) for application to very large-scale generalised assignment and other combinatorial optimisation problems is presented in this paper. The new TS applies dynamic oscillation of feasible verses infeasible search and neighbourhood sample sizes that vary throughout the solution process. The dynamic oscillation and neighbourhood sample sizes are controlled by the success of the search as the solution progresses, to allow a faster increase in solution quality per unit time. Application of the TS to three types of randomly generated very large-scale generalised assignment problem instances was performed for sizes of up to 50 000 jobs and 40 agents. The new TS gave superior solutions to existing versions on all nearly occasions, given a fixed CPU time. For a fixed solution quality, the best of the existing versions required 1.5–3 times as much CPU time.

Scope and purpose

Very large-scale generalised assignment problems of several thousand jobs have important real-life applications such as vehicle routing, project planning, computer network design and sugar cane supply management. Existing heuristic techniques for the NP-hard problem are designed for medium-size problem instances of a few 100 jobs, since CPU time or search space significantly increases with problem size. This paper puts forward a new version of tabu search designed for very large generalised assignment and other large combinatorial optimisation problems. The new tabu search incorporates dynamic oscillation and neighbourhood sample sizes. These are controlled by the success of the search as the solution progresses, to allow a faster increase in solution quality per unit time. The results reported in the paper for problem sizes of up to 50 000 jobs and 40 agents, show a considerable reduction in CPU time over existing versions of tabu search to reach a desired solution quality.

Keywords: Generalised assignment problem; Tabu search; Local search; Oscillation

* Tel.: + 61-7-3214-2206; fax: + 61-7-3214-2340.
E-mail address: andrew.higgins@tag.csiro.au (A.J. Higgins).

0305-0548/01/$ - see front matter
PII: S0305-0548(00)00024-1

1. Introduction

The generalised assignment problem (GAP) is a popular NP-hard combinatorial optimisation [1] problem that involves assigning n jobs to m agents where each job can only be assigned to one agent. All jobs must be assigned and the agents are subject to capacity constraints. The objective is to maximise the profit or minimise the cost of allocating the jobs to the agents. The GAP and its extensions have many practical applications in vehicle routing [2], resource allocation [3], sugar cane harvest scheduling [4] and so on. The mathematical formulation for the GAP is as follows:

$$\text{Maximise} \quad Z = \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} x_{ij} \tag{1}$$

subject to

$$\sum_{i=1}^{n} a_{ij} x_{ij} \leqslant b_j, \quad j \in \{1, \ldots, m\}, \tag{2}$$

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

$$x_{ij} \in \{0, 1\}, \quad i \in \{1, \ldots, n\}, \quad j \in \{1, \ldots, m\}, \tag{4}$$

where x_{ij} is the decision variable, $x_{ij} = 1$ if job i is assigned to agent j, 0 otherwise, p_{ij} is the profit of assigning job i to agent j, a_{ij} is the resource required for assigning job i to agent j, and b_j is the resource capacity of agent j.

Since the GAP is NP-hard and optimal solutions can only be found for small problem instances, research has focused on heuristic solution techniques. Many of the approximation solution methods for the GAP are based on some form of relaxation, whether Lagrangian, surrogate or LP relaxation [5–7]. These methods are more suited to medium-size problem instances (approximately 200 jobs). For example, with LP relaxation, the number of LP decision variables is $m*n$ which is 2 000 000 for the largest problem in this paper. A genetic algorithm was applied [8] to solve the GAP for problem instances of up to 200 jobs and 20 agents. The reported solution time for that size instance was over one CPU hour. Memory and CPU requirements increase substantially for large size problem instances since a population of solutions must be manipulated at each generation of the genetic algorithm solution process.

Many of the recent heuristic solution techniques applied to the GAP have been based upon different versions of local search [9]. The paper by Amini and Racer [10] combines the greedy heuristic with a local search refinement that terminated when no further improvement was possible. Chain moves were allowed in the search [10] to overcome some local optimal solutions. Results were reported for problem instances with up to 200 jobs and 20 agents for which solution time was about 30 s. Tabu search (TS) [11,12] and simulated annealing were applied to the GAP [13] and multi-level GAP [14] for problem instances of up to 60 jobs and 10 agents. Whilst the problem sizes were not large, TS and simulated annealing can be applied to much larger sizes without an explosion in the memory requirements or CPU time to obtain a good solution.

This paper applies a new and existing versions of TS to very large GAPs with tight resource constraints. The new version of TS was motivated by a problem found in the Australian Sugar

Industry [4] that required the optimisation of harvest dates of farm paddocks to maximise profitability. In such a problem, the farm paddocks were jobs and weeks of harvest were agents. This resulted in a problem size of about 150 000 jobs and 12 agents. Whilst the problem included more complex constraints than the GAP, the new TS was applied in exactly the same way.

The existing versions of TS applied in this paper vary in terms of how infeasibility search is handled. One method is oscillation that allows the TS to swap between feasible and infeasible search spaces. However, as the TS proceeds over time, the benefits of search through feasible versus infeasible space will vary. The version of oscillation for the new TS in this paper is dynamic in which the weighting between feasible and infeasible space varies to be at the best level at any stage of the solution process. This concept is also applied to have the best weighting of neighbourhood sample sizes at any stage of the solution process. Having the dynamic neighbourhood sample sizes and oscillation reduces the amount of wasted search throughout the solution process and thus reduces the solution time to find the desired solution quality.

2. Tabu search

The general tabu search heuristic is based on the establishment of moves so as to transform a current solution to one of the neighbouring. The tabu search escapes local optimal solutions by allowing up-hill (non improving) moves to be performed when no down-hill (improving) moves are available. At each iteration of the tabu search, the neighbourhood (or a sample of it) is searched for which the best non-tabu move found in the search is applied. A move is tabu if it is one of the *TL* (tabu list) most recent moves applied. If this tabu list size *TL* is too small, the heuristic will cycle through a series of solutions. The tabu status is over-ridden if the solution satisfies an aspiration criteria function. Depending on the problem at hand, the general tabu search can be extended so as to promote various forms of diversification and intensification.

In this paper, two neighbourhoods were applied to the GAP as follows:

Neighbourhood 1: *Insert move*. If $x_{ij} = 1$, then let $x_{ij} = 0$, $x_{ij^{\#}} = 1$ where $j \neq j^{\#}$.

Neighbourhood 2: *Swap move*. If $x_{ij} = 1$ and $x_{i^{\#}j^{\#}} = 1$ where $i \neq i^{\#}$ and $j \neq j^{\#}$, then let $x_{ij} = 0$, $x_{ij^{\#}} = 1, x_{i^{\#}j} = 1, x_{i^{\#}j^{\#}} = 0$.

The two neighbourhoods complement one another during the TS. Swap moves will be applied more frequently when it is difficult to improve the solution due to constraints (2) being extremely tight.

Unlike previous applications of TS for small- to medium-size GAPs [13,14], for the large problem sizes described in this paper (up to $n = 50\,000, m = 40$), a full neighbourhood search is far too inefficient due to the size being $1.25{*}10^9$ possible swap moves ($n^2/2$). From this, a sample of the neighbourhoods was chosen [15], with a combined sample size *NS*. Different values of *NS* were experimented with for 10 instances of each problem size to find a value that produced the overall best results ($NS = 1600$). This sample size of the neighbourhood produced superior results over much larger sample sizes, for the following reason: say a small sample size quickly finds a solution

that is 0.1% better than the best so far. A sample size of 10 times the above size may only find a solution 0.15% better than the best so far, but gives rise to considerable wasted time in the search.

Since a sample of the neighbourhood is searched at each iteration of the TS, TL had to be set to at least 1000 (for smaller problem sizes up to $n = 2000$, and larger still for $n > 2000$) to significantly improve the final solution. This was assessed by experimenting with different values of TL for 10 instances of each problem size. The solution quality declined for very large TL due to the time required checking the tabu list. One may argue then, for very large problem sizes, a tabu list is not required. In this case, the heuristic would be called a dynamic local search, which incorporates some of the ideas of TS.

The search is diversified by replacing the current solution with the best solution so far after a predefined number of iterations, $\phi^{\#}$, have passed without any improvement upon the best solution. The value of $\phi^{\#}$ was found to be dependent on the neighbourhood sample size. In the case where infeasible solutions are allowed, $\varphi^{\#}$ is the predefined maximum number of iterations for which a solution can be infeasible, after the first feasible solution has been found. As with NS, different combinations of $\phi^{\#}$ and $\varphi^{\#}$ were tested on several instances of each problem size to achieve the values that produce the best overall results. For the sample size defined above, the best values found of $\phi^{\#}$ and $\varphi^{\#}$ were 10 and 50, respectively.

An initial solution for the TS is generated by allocating each job to the agent that maximises Z but does not violate constraints (2). If the job cannot be allocated to an agent without violating constraints (2), the job is allocated to the agent that minimises the amount to which this constraint is violated ($\delta = \sum_{j=1}^{m} \max(\sum_{i=1}^{n} a_{ij}x_{ij} - b_j, 0)$). The following algorithm is commenced with setting the current solution CS to this initial solution.

Set $BS := 0$ (initialises the best solution)

Repeat
  Set $\phi := 0$
  Set $\varphi := 0$
  Repeat
    Obtain a sample (size NS) of moves from the neighbourhood of CS
    Set $CS :=$ the solution given the best non tabu move in the sample (using the version of TS below)
    Add this move to the tabu list
    If $CS > BS$ and $\delta = 0$ **then**
      Set $BS := CS$ and $CFS := CS$ (keeps record of the best and current feasible solution)
      Set $\phi := 0$
    ElseIf $\delta = 0$ and $CS \leqslant BS$ **then**
      Set $\phi := \phi + 1$
    End_If
    If $\delta > 0$ **then**
      Set $\varphi := \varphi + 1$
    Else
      Set $\varphi := 0$
    End_If

Until $\phi = \phi^{\#}$ or $\varphi = \varphi^{\#}$ or $CFS > CS$
Set $CS := BS$ and $CFS := BS$
Until CPU time > upper limit

Along with $\phi^{\#}$ and $\varphi^{\#}$ in the above algorithm, intensification is also performed if the current solution is infeasible and is not likely to be made feasible (when considering a linear trend of δ) within $\varphi^{\#}$.

2.1. Versions of tabu search

Four different versions of TS were tried out of which the first three are existing in the literature. The fourth is the new version, dynamic oscillation and neighbourhood sample sizes.

(1) *Infeasible solutions not allowed*: In the case where a move is not allowed to violate constraints (2) after the first feasible solution is found, the best move is the one which gives the largest Z. When this version is used, $\varphi^{\#}$ in the above algorithm does not apply.

(2) *Infeasibility controlled by rules*: An infeasible solution is allowed for which the best move from the sample is selected in one of the following ways: (i) if $\delta = 0$ (for CS) and the best feasible move produces a solution better than CS then select it; (ii) if $\delta = 0$ and the best feasible move produces a worse solution than CS and the best infeasible move produces a better solution than CS then select the best infeasible move; (iii) if $\delta = 0$ and both the best feasible and infeasible moves produce a worse solution than CS, then select the best feasible move; and (iv) if $\delta > 0$ then select the move which leads to the greatest reduction in δ. This version has been applied by Laguna et al. [14] for the multilevel GAP.

(3) *Infeasibility through oscillation*: Given Z' and δ' are the objective function value and a measure of violation of constraints (2) for a solution produced by a move from the neighbourhood of CS, the fitness of this move is calculated by

$$Z' - w1*\delta' \quad \text{if } \delta = 0,$$

$$Z' - w2*\delta' \quad \text{if } \delta > 0.$$

Here, the search is permitted through infeasible space, since δ can be > 0. The constant weights $w1$ and $w2$ are used to give the best weighting between Z' and δ'. The move with the best fitness is selected. The advantage over version 2 is that both Z' and δ' are considered simultaneously. For example, an excellent move with a small δ' (which is easily made feasible with little change to Z') will be selected here. In version 2, a much poorer move would be selected instead. A similar fitness function was applied to the multiple facility loading [16] and multi-dimensional knapsack problems [17].

(4a) *Infeasibility through dynamic oscillation*: This new version is an extension of version 3 in that it varies the weighting of Z' with respect to δ', to produce the best feasible solution gains per CPU time. The fitness of a move is

$$Z' - (w1 + w3*(mf - mif))*\delta' \quad \text{if } \delta = 0,$$
$$Z' - w2*\delta' \quad \text{otherwise,} \qquad \text{if } \delta > 0,$$

The value *mf* is the average increase in Z per move over the past *pmf* moves that have transformed a feasible solution into another. The value *mif* is the average increase in Z over the past *pmif* moves that have transformed a feasible solution into an infeasible one, infeasible to feasible and infeasible to infeasible. The move with the best fitness in the sample is selected.

This method has a fixed weight $w1$ and variable weight $w3*(mf - mif)$ on how much δ' affects the fitness value. For example, say that at a given stage of the solution process, the search through feasible space was giving far superior improvements in Z than that through infeasible space. This will cause $(mf - mif)$ to be large which will make the fitness of a move very sensitive to δ' relative to Z'. The likelihood of an infeasible move being selected will then be reduced. Selection of values $w1$–$w3$ was done with the same experimentation as for NS. For $w1$–$w3$ these were 0, 0.004 and 0.1, respectively.

(4b) *Dynamic neighbourhood and oscillation*: This second part of the new version extends (4a) to dynamically refining the sample size of neighbourhood 1 relative to 2. The purpose is to keep the best neighbourhood sample size of insert moves relative to swap moves throughout the solution process. For example, at a given stage of the solution process, insert moves, on average, may give a much larger gain in Z than swap moves. It would then be ideal for the neighbourhood sample size for insert moves to be much larger than that of swap moves. The sample size of neighbourhood 1 relative to 2 is dynamically calculated as follows:

Out of the last *bm* moves chosen in the TS, let

$rm1$ = number of times from neighbourhood 1,

$rm2$ = number of times from neighbourhood 2, $bm = rm1 + rm2$.

Let

$as1$ = average sample size of neighbourhood 1 during the last *bm* moves of the TS,

$as2$ = average sample size of neighbourhood 2 during the last *bm* moves of the TS,

$su1 = rm1/as1$, i.e. average success factor of neighbourhood 1 over the last *bm* moves of the TS,

$su2 = rm2/as2$,

$$\text{Sample size of neighbourhood 1} = \min\left(\max\left(0.1, \frac{su1}{su1 + su2}\right), 0.9\right)NS,$$

$$\text{Sample size of neighbourhood 2} = NS - \text{Sample size of neighbourhood 1.} \tag{5}$$

From (5) above, the sample size of neighbourhoods 1 and 2 is constrained to be between 10 and 90% of the NS. This prevents one neighbourhood sample from being too dominant and not allowing the other to increase in size. Through experimenting, the value of *bm* was set to 100.

3. Numerical experiments

Comparisons were made between each of the existing and new TS versions on a Pentium II 400 when programmed using Fortran 90. The TS versions were firstly applied to previously used GAP

test problem instances [8] with known optimal solutions for up to 60 jobs and 10 agents. All versions of the TS found the optimal solution or was within 0.1% of the optimal while requiring less than 1 s of CPU time. To make sufficient comparisons of the TS versions, much larger problem instances were generated randomly with sizes ranging from 2000 jobs, 20 agents to 50 000 jobs, 40 agents. The latter results in a problem of 2 million decision variables.

3.1. Test problems

The test problem instances used were generated by three different methods [18] as follows.

1. Using a uniform distribution, choose a_{ij} randomly between 5 and 25 and p_{ij} between 1 and 40. Let $b_j = \eta \sum_{i=1}^{n} a_{ij}/m$, where constant η is chosen so as to achieve the desired tightness in constraints (2)
2. Using a uniform distribution, choose a_{ij} randomly between 1 and 100. Let $p_{ij} = 111 - a_{ij} + \lambda$ where λ is random between -10 and 10. As with method $1, \ldots, b_j = \eta \sum_{i=1}^{n} a_{ij}/m$.
3. Given that $\lambda 1$ and $\lambda 2$ are random numbers between 0 and 1, let $a_{ij} = 1 - 10\lambda 1$, $p_{ij} = 1000/a_{ij} - 10\lambda 2$, $b_j = \max(0.8 \sum_{i=1}^{n} a_{ij}/m, \max_i(a_{ij})) * \eta$, where constant η is selected to achieve the desired tightness in constraints (2).

For each of the above test problem sets and each problem size, 10 random problem instances were generated. The value of η was selected for each problem set and size, to have constraints (2) as tight as possible but at the same time allowing a feasible solution to be found fairly quickly.

4. Results

The versions of the TS were firstly assessed by comparing the average Z, given that the TS was terminated 10 min after the initial solution was found. For the existing versions of TS, the ratio of neighbourhood sample sizes was set at 3 : 1 for neighbourhood 1 versus neighbourhood 2. This ratio was determined by experimenting with different combinations for 10 instances of each problem size, and picking the ratio that gave the best overall results. In Table 1, which presents the results in terms of average difference between Z and Z_{best} from the best performing version ($= 1 - Z/Z_{\text{best}}$ expressed as a percentage), the new version of TS (dynamic neighbourhoods and oscillation) gave average results superior to the best of the existing TS versions on all but one occasion. The new TS version (infeasibility through dynamic oscillation) outperformed (infeasibility through oscillation) on 69% of problem instances. The TS version (infeasibility through oscillation) outperformed the other versions in literature on nearly all occasions. The performance of the TS versions did vary with the method used to generate the problem set, with the new TS versions performing far superior to existing TS versions on set 1.

To assess solution time differences, the new TS version (4b) (dynamic neighbourhoods and oscillation) was run with a CPU time of 2 min and the best solution recorded. TS versions 1 (infeasible solutions not allowed) and 3 (infeasibility through oscillation) were then run until a solution was reached which was at least as good as that recorded for TS version (4b). Once the

Table 1
Average difference (%$*10^2$) from best version

Problem set	Size		Tabu search version[a]				
	n	m	1	2	3	4a	4b
1	2000	20	19.15	12.04	7.43	3.39	0[b]
	10 000	20	38.62	15.09	8.87	4.43	0
	10 000	40	33.29	21.90	20.76	22.88	0
	50 000	40	1.96	2.09	2.20	2.23	0
2	2000	20	2.08	0	0.28	0.47	0.33
	10 000	20	4.59	0.67	0.74	0.28	0
	10 000	40	7.36	2.45	1.43	0	0.09
	50 000	40	2.79	2.11	1.57	1.75	0
3	2000	20	0.35	0.27	0.09	0.16	0
	10 000	20	0.27	0.21	0.01	0.01	0
	10 000	40	0.70	0.80	0.06	0.11	0
	50 000	40	4.56	0.77	0.09	0.02	0

[a]Tabu search version
1 Infeasible solutions not allowed.
2 Infeasibility controlled by rules.
3 Infeasibility through oscillation.
4a Infeasibility through dynamic oscillation (new).
4b Dynamic neighbourhoods and oscillation (new).
[b]The best version will have a value of 0.

solution was reached, the CPU time required to obtain it was recorded. This was done for 10 random problem instances within each problem size and set. In Table 2, the best TS version in the literature (infeasibility through oscillation) generally required between 3 and 6 min to achieve the same solution quality as the new TS version (dynamic neighbourhoods and oscillation) when the latter was allowed to run with a CPU time of 2 min. The TS version (infeasible solutions not allowed) required up to 40 min of CPU time with over half of the problem instances requiring at least 15 min.

Summary

This paper has presented a new TS for the GAP and other combinatorial optimisation problems. The new TS implemented dynamic oscillation and neighbourhood sample sizes that makes best use of infeasible verses feasible space and relative sample size of neighbourhoods as the solution progresses. Applying three different methods, randomly generated large-scale GAP test problem instances of up to 50 000 jobs and 40 agents were used to compare the new version of TS with three existing versions in the literature.

Table 2
Average CPU time (minutes) required to achieve solution quality of TS version 4b[a]

Problem	Size		Tabu search version[b]	
set	n	m	1	3
1	2000	20	21.0	5.3
	10 000	20	17.1	3.7
	10 000	40	3.9	2.8
	50 000	40	6.8	4.4
2	2000	20	39.2	28.8
	10 000	20	8.6	5.1
	10 000	40	17.6	2.9
	50 000	40	30.0	4.1
3	2000	20	4.4	3.3
	10 000	20	7.6	2.9
	10 000	40	21.8	5.0
	50 000	40	24.5	4.7

[a]New TS version (4b) run for 2 min.
[b]Tabu Search version
1 Infeasible solutions not allowed.
3 Infeasibility through oscillation.

When the TS versions were run for 10 min of CPU time, the new TS with dynamic neighbourhoods and oscillation, produced superior average results to existing TS versions in the literature on nearly all occasions. When the new TS was allowed to run with a CPU time of 2 min, the best existing TS version required an average of 3 and 6 min to achieve the same solution quality.

The TS of this paper can be applied to larger problems (e.g., 500 000 jobs and 400 agents) and is only bounded by the available computer memory and the desired solution quality within a predefined CPU time frame. Since a sample of the neighbourhood is taken and the new features of the TS in this paper do not slow down with increased problem size, the CPU time required for each iteration does not need to increase with problem size.

References

[1] Fisher ML, Jaikumar R, Van Wassenhove L. A multiplier adjustment method for the generalised assignment problem. Management Science 1986;32:1095–103.
[2] Fisher ML, Jaikumar R. A generalised assignment heuristic for vehicle routing. Networks 1981;11:109–24.
[3] Foulds LR, Wilson JM. A variation of the generalised assignment problem arising in the New Zealand dairy industry. Annals of Operations Research 1997;69:105–14.

[4] Higgins AJ. Optimising cane supply decisions within a sugar mill region. Journal of Scheduling 1999;2:229–44.
[5] Cattrysse DG, Van Wassenhove LN. A survey of algorithms for the generalised assignment problem. European Journal of Operational Research 1992;60:260–72.
[6] Trick M. A linear relaxation heuristic for the generalised assignment problem. Naval Research Logistics 1992;39:137–51.
[7] Lorena LAN, Narciso MG. Relaxation heuristics for a generalised assignment problem. European Journal of Operational Research 1996;91:600–10.
[8] Chu PC, Beasley JE. A genetic algorithm for the generalised assignment problem. Computers and Operations Research 1996;24:17–23.
[9] Papadimitriou CH, Steiglitz K. Combinatorial optimisation: algorithms and complexity.. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[10] Amini MM, Racer M. A hybrid heuristic for the generalised assignment problem. European Journal of Operational Research 1995;88:343–8.
[11] Glover F. Tabu search: a tutorial. Interfaces 1990;20:74–9.
[12] Glover F. A user's guide to tabu search. Annals of Operations Research 1993;41:3–28.
[13] Osman IH. Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. OR Spektrum 1995;17:211–25.
[14] Laguna M, Kelly JP, Velarde JLG, Glover F. Tabu search for the multilevel generalised assignment problem. European Journal of Operational Research 1995;82:176–89.
[15] Semet F, Taillard E. Solving real life vehicle routing problems efficiently using tabu search. Annals of Operations Research 1993;41:469–88.
[16] Mazzola JB, Schantz RH. Multiple facility loading under capacity based economics of scope. Naval Research Logistics 1997;44:229–56.
[17] Hanafi S, Freville A. An efficient tabu search approach for the 0–1 multidimensional knapsack problem. European Journal of Operational Research 1998;106:659–75.
[18] Racer M, Amini MM. A robust heuristic for the generalised assignment problem. Annals of Operations Research 1994;50:487–503.

Andrew Higgins obtained his Ph.D. from the Queensland University of Technology, Australia, in 1996 and is currently working at the CSIRO Tropical Agriculture, Brisbane, as an operations research scientist. His current research interests include applying operations research techniques to complex industry problems and extending meta-heuristics for large scale integer programming problems.