J. Opl Res. Soc. Vol. 44, No. 7, pp. 673-679
Printed in Great Britain. All rights reserved

0160-5682/93 $9.00+0.00
Copyright © 1993 Operational Research Society Ltd

Categorized Assignment Scheduling: a Tabu Search Approach

ABRAHAM P. PUNNEN and Y. P. ANEJA
Faculty of Business Administration, University of Windsor, Canada

The assignment problem (AP) and bottleneck assignment problem (BAP) are well studied in operational research literature. In this paper we consider two related problems which simultaneously generalize both AP and BAP. Unlike AP and BAP, these generalizations are strongly NP-complete. We propose two heuristics to solve these generalized problems: one based on a greedy principle and the other based on tabu search. Computational results are presented which show that these heuristics, when used together, produce good quality solutions. Our adaptation of tabu search also gives some new insight into the application of tabu search on permutation problems.

Key words: assignment, scheduling, heuristics, tabu search, greedy algorithm

INTRODUCTION

Let $I = \{1, 2, \ldots, n\}$ be a finite set and Π be the set of all permutations defined on I. Any permutation $\pi \in \Pi$ is identified as an n vector $(\pi(1), \pi(2), \ldots, \pi(n))$. Let $C = (C_{ij})$ be an $n \times n$ matrix and $S_1, S_2, \ldots, S_p$ be a given partition of I. For any $\pi \in \Pi$, let

$$f_k(\pi) = \sum_{j \in S_k} C_{j\,\pi(j)}$$

be the total 'cost' corresponding to π with respect to S_k. Then the categorized assignment scheduling problem can be defined as:

$$\text{CAS1: } \min_{\pi \in \Pi} \max_{1 \leqslant k \leqslant p} f_k(\pi).$$

Another problem closely related to CAS1 is obtained by interchanging 'max' and 'Σ' in CAS1, and can be defined as:

$$\text{CAS2: } \min \sum_{k=1}^{p} \max_{j \in S_k} \{C_{j\pi(j)}\}.$$

For $p = 1$, CAS1 reduces to the classical assignment problem and CAS2 reduces to the bottleneck assignment problem[1]. Likewise for $p = n$, CAS2 reduces to the classical assignment problem and CAS1 reduces to the bottleneck assignment problem. Thus, both CAS1 and CAS2 simultaneously generalize the classical assignment problem and the bottleneck assignment problem.

A physical interpretation of problems CAS1 and CAS2 can be given as follows. Suppose that n jobs are to be assigned on n machines in such a way that each job must be assigned to some machine and each machine can do exactly one job. Further, assume that the jobs are partitioned into different categories. Now if jobs in the same category must be processed in series, but different category jobs can be processed in parallel, then the CAS1 objective function measures the processing time. Similarly, if jobs in the same category can be processed in parallel, but jobs in a different category can be started only after all the jobs in the currently processing category are finished, then the CAS2 objective function measures the processing time. Several real-life applications of CAS1 and CAS2 are mentioned in the literature.[2,3]

The problems, CAS1 and CAS2, have been studied by Seshan[2], Aggarwal *et al.*[3] and by Richey and Punnen[4]. Seshan[2] proposed two branch-and-bound algorithms to solve CAS1 and CAS2,

Correspondence: Y. P. Aneja, Faculty of Business Administration, University of Windsor, Windsor, Ontario, Canada N9B 3P4

but no computational results indicating the behaviour of the algorithms are available. Aggarwal *et al.*[3] addressed particular cases of CAS1 and CAS2 and mentioned several real-life applications. Richey and Punnen[4] studied the complexity aspects of CAS1 and CAS2. They showed that both CAS1 and CAS2 are strongly NP-complete. This rules out (modulo $P = NP$) the existence of even a Pseudo-polynomial algorithm for solving CAS1 and CAS2[5]. Once a problem with good real-life applications is identified to belong to the class of NP-complete problems, a natural attempt is to develop efficient heuristics to solve the problem approximately. In the case of CAS1 and CAS2, to our knowledge, such a study has not been initiated so far.

In this paper we develop two heuristics to solve CAS1 and CAS2. When used together, these heuristics produce good quality approximate solutions. The first heuristic is a constructive one called MINMAX_GREEDY. Starting with the solution obtained by MINMAX_GREEDY, we use a tabu search heuristic called TABU_CAS to generate improved solutions. We also present results of extensive computational experiments conducted using the proposed heuristics. In the paper, we focus our attention on CAS1. The algorithm for CAS2 follows exactly on similar lines, with appropriate modifications and, therefore, is not discussed.

THE MINMAX_GREEDY HEURISTIC

We now discuss our constructive heuristic MINMAX_GREEDY to solve CAS1. The algorithm can be considered as an extension of the LPT heuristic developed for the generalized 3-partition problem[6] and the multi-processor scheduling problem[7]. At any iteration, given a partial assignment, the algorithm picks the set S_k with smallest total cost corresponding to the given partial assignment. It then assigns i to j such that (i) $i \in S_k$; (ii) neither i nor j have been assigned in the partial assignment; and (iii) c_{ij} is as small as possible. A formal description of the algorithm is given below.

Algorithm MINMAX_GREEDY

```
Step 0: (initialization)
    Let $\alpha_1^k \leqslant \alpha_2^k \leqslant \ldots \leqslant \alpha_{n|s_k|}^k$ be an ascending arrangement of $C_{ij}$, $i \in S_k$ and $1 \leqslant j \leqslant n$, $1 \leqslant k \leqslant p$.
    For each $\alpha_t^k$ let $I(\alpha_t^k)$ and $J(\alpha_t^k)$ be such that $\alpha_t^k = C_{I(\alpha_t^k), J(\alpha_t^k)}$
    Let flag1$(i) = 0$, for $1 \leqslant i \leqslant n$ and flag2$(j) = 0$, for $1 \leqslant j \leqslant n$.
    (if $i$ is assigned to $j$ flag1$(i) = j$ and flag2$(j) = i$)
    Set
                $F_k = 0, k = 1, \ldots, p$
    ($F_k$ represents the 'cost' of the 'partial' assignment corresponding to the set $S_k$)
    ASSIGN$(k) = 0$, $k = 1, \ldots, p$
    (ASSIGN$(k)$ represents the number of assignments made in set $S_k$)
    TOTAL_ASSIGN = 0
    (TOTAL_ASSIGN refers to the total number of assignments made so far)
                $\beta_k = 1, k = 1, \ldots, p$
    end set

Step 1: Choose $r$ such that
                $F_r = \min_{k \in X} \{F_k\}$.
    Where
                $X = \{k / \text{ASSIGN}(k) < |S_k|, 1 \leqslant k \leqslant p\}$.
    (Ties are broken by choosing $r$ with the largest value $|S_r|$. Further ties are broken arbitrarily)
```

```
Step 2: If flag1$(I(\alpha^r_{\beta_r}))$ or flag2$(J(\alpha^r_{\beta_r}))$ is not zero go to Step 3
        flag1$(I(\alpha^r_{\beta_r})) = J(\alpha^r_{\beta_r})$ and flag2$(J(\alpha^r_{\beta_r})) = I(\alpha^r_{\beta_r})$
    $\beta_r = \beta_r + 1$
    ASSIGN(r) = ASSIGN(r) + 1
    TOTAL_ASSIGN = TOTAL_ASSIGN + 1
    $F_r = F_r + \alpha^r_{\beta_r}$.
    If TOTAL_ASSIGN = N Then STOP
    /* $(i, \text{flag1}(i)), i = 1, \ldots, n$ is the MINMAX_GREEDY solution */
    else go to Step 1

Step 3: $\beta_r = \beta_r^{+1}$ go to Step 2.
```

It is easy to verify that the algorithm MINMAX_GREEDY terminates with a feasible assignment. The complexity analysis of the algorithm can be described as follows. The construction of the ascending arrangement $\alpha_1^k \leqslant \alpha_2^k \leqslant \ldots \leqslant \alpha^k_{n|S_k|}$ can be done in $O(|S_k| \log |S_k|)$ time. Repeating this for p sets yields a complexity of

$$\sum_{k=1}^{p} O(|S_k| \log |S_k|) = O(n^2 \log(\max_{1 \leqslant k \leqslant p} |S_k|)) \leqslant O(n^2 \log n).$$

It can be verified that the rest of the computations in the algorithm take only $O(n^2)$ time. Thus, the worst case complexity of the algorithm MINMAX_GREEDY is $O(n^2 \log n)$.

We will discuss the quality of the solution produced by MINMAX_GREEDY later, in the section on computational results. Let us now consider our tabu search heuristic.

TABU SEARCH HEURISTIC

Tabu search is a general heuristic procedure developed by Glover[8-10] for solving hard combinatorial optimization problems. We adapt the following definitions from References 8 and 11. A *move* is a transition from one permutation to another by interchanging locations of two elements i and j. The unordered pair (i, j) is called the *attribute* of this move. The *value of a move* is the difference between the objective function values after and before the move. If the value of a move is negative, it is called an *improvement move* otherwise it is a non-improvement move. Let π be any permutation and π_{ij} be a permutation obtained from π by interchanging locations of $i \in S_{k_i}$ and $j \in S_{k_j}$. That is $\pi_{ij}(r) = \pi(r)$ if $r \neq i, j$ and $\pi_{ij}(i) = \pi(j)$ and $\pi_{ij}(j) = \pi(i)$. Let $f_k(\pi) = \Sigma_{t \in S_k} C_{t\pi(t)}$. Then $f_k(\pi_{ij})$, $1 \leqslant k \leqslant p$, can be obtained in $O(1)$ time using the formula

$$f_k(\pi_{ij}) = \begin{cases} f_k(\pi) & \text{if } k \neq k_i, k_j, \\ f_k(\pi) - C_{i\pi(i)} - C_{j\pi(j)} + C_{i\pi(j)} + C_{j\pi(i)} & \text{if } k = k_i = k_j, \\ f_k(\pi) - C_{i\pi(i)} + C_{i\pi(j)} & \text{if } k = k_i \neq k_j, \\ f_k(\pi) - C_{j\pi(j)} + C_{j\pi(i)} & \text{if } k = k_j \neq k_i. \end{cases}$$

Thus, the value of a move, $\Delta(\pi, i, j)$, from π to π_{ij} can be computed in $O(p)$ time using

$$\Delta(\pi, i, j) = \max_{1 \leqslant k \leqslant p} f_k(\pi_{ij}) - \max_{1 \leqslant k \leqslant p} f_k(\pi). \tag{1}$$

Further, it may be noted that $\Delta(\pi, i, j)$ can be updated using a more refined data-structure[12] in $O(\log p)$ time.

The *best move* is a move in which the value of the move is as small as possible. The best move need not be an improving move. A kth best move is a move in which the value of the move is kth smallest (counting multiplicity) among all admissible moves (defined later). During the course of the algorithm, a BEST_LIST of pairs (i, j) which corresponds to the k best moves,

$1 \leqslant k \leqslant$ BEST_SIZE (a given parameter) is maintained and updated. Similarly, a TABU_LIST of pairs (i, j) of length TABU_SIZE (a given parameter) is maintained and updated during the algorithm. If a pair (i, j) belongs to the TABU_LIST, at a prescribed iteration, the move with attribute (i, j) is inadmissible. However, such an inadmissible move becomes admissible if it results in a solution which improves the best available objective function value so far. This is the *aspiration level* criterion. Generally in permutation problems, TABU_LIST is maintained as a circular list of length TABU_SIZE.[11] In this type of data structure, to identify whether a pair (i, j) is in the TABU_LIST, one has to scan through the entire TABU_LIST and this takes O(TABU_SIZE) time. We use here a different data structure for maintaining TABU_LIST by which the tabu status of a pair can be verified in $O(1)$ time. For this we keep an iteration counter ITER_A, an $n \times n$ matrix TABU_MAT and a number CTSIZE which indicates the current TABU_SIZE. ITER_A, CTSIZE and all elements of TABU_MAT are initialized to 0. After each move, ITER_A is increased by 1 if the move is an ordinary move (i.e. a move not resulting from passing the aspiration criterion) and ITER_A remains unchanged otherwise. Similarly, after each move CTSIZE is increased by one if it is an ordinary move and CTSIZE < TABU_SIZE. After every ordinary move with attribute (i, j) TABU_MAT is updated by putting TABU_MAT(i, j) = ITER_A. Now, at any iteration, a pair (i, j) is tabu if and only if ITER_A − TABU_MAT(i, j) < CTSIZE. Thus, the tabu status of a pair (i, j) can be verified in $O(1)$ time.

TABU SIZE, BEST SIZE and cycling

The performance of a tabu search algorithm depends on several parameters. One of the most important parameters is the TABU_SIZE—the length of the TABU_LIST. If it is too small, then the algorithm is very likely to enter into cycling. If TABU_SIZE is too large, then it will restrict the domain of search considerably and good solutions are likely to be skipped. Thus, it is desirable to keep TABU_SIZE small and still somehow prevent cycling. For this purpose we introduce the BEST_LIST and the parameter BEST_SIZE. It may be noted that if cycling occurs, it does so only after the algorithm encounters the first local minimum. From a local minimum, the algorithm either makes a 'hill climbing' move by increasing the objective function value or travels through alternate local optimal solutions. If the algorithm makes an 'uphill' move, then the possibility of cycling arises when it makes 'down hill' moves in later iterations. So during this 'down hill' phase we introduce some 'randomness' using the BEST_LIST. Instead of making the 'best move' at this stage we move to a point in the BEST_LIST randomly. A similar kind of randomness is incorporated while the algorithm passes through alternate local optimum solutions. Since BEST_LIST is a list of 'good' moves, it does not worsen the objective function value too much and the resulting randomness is helpful in preventing cycling.

Diversification of search

After having gone through a prescribed number (say MAX_ITER) of iterations, it is desirable to restart the search through a different path. To achieve this, a generally-used technique is to maintain a long term memory which 'remembers' previous paths. In permutation problems, where pair-wise exchange is used to define a move, a popular long term memory used is through an $n \times n$ matrix,[11] say, LTM. LTM is initialized to a zero matrix and the value of the (i,j)th element of LTM is increased by one when a move with attribute (i,j) is performed. Then after MAX_ITER iterations, the original cost matrix C is replaced by $C + \mu$LTM, where μ is a prescribed parameter and the search is restarted from an appropriate solution. In addition to the long term memory, we propose another way to diversify search paths using the BEST_LIST. By choosing different values for BEST_SIZE, the search path gets diverted.

We now give a formal description of our tabu search algorithm for solving CAS1.

Algorithm tabu-CAS1

Step 0: (initial solution)

READ Problem data (The matrix C, the partition vector, number of partitions, size of the permutation and the parameter μ).

```
Obtain an initial starting solution using the MINMAX_GREEDY heuristic.

Search:
Step 1. Read algorithm parameters (TABU_SIZE, BEST_SIZE and MAX_ITER) and initialize
        long term memory.
Step 2. Repeat
        • construct a BEST_LIST of admissible moves of size BEST_SIZE (an admissible move
          is a non-tabu move or a tabu move which passes the aspiration criterion);
        • if a local minimum has not been attained in an earlier iteration or the best move is an
          'uphill' move, make a best move. Else make a random move with attribute in
          BEST_LIST;
        • update the TABU_LIST, long term memory and best solution found so far (if
          necessary);
        until MAX_ITER iterations are over.
Step 3. Diversification of search path
        • Choose any of the following options:
          (1) Restart from the previous MINMAX_GREEDY solution.
          (2) Restart from the best solution found so far.
          (3) Restart from the last solution in the previous step.
          (4) Use MINMAX_GREEDY with cost matrix $C + \mu \times$ LTM and restart from the
              resulting solution.
          (5) Stop the algorithm and output the best solutions found so far.
        • Choose one or more of the following options to modify algorithm parameters:
          (1) (Clear TABU_LIST) read new TABU_SIZE.
          (2) Read new BEST_SIZE.
          (3) Read new MAX_ITER.
          (4) Replace $C$ by $C + \mu \times$ LTM.
Step 4. Go to Step 2
End Search
```

COMPUTATIONAL RESULTS

In this section, we present results of some preliminary computational experiments conducted using algorithms MINMAX_GREEDY and TABU_CAS1. Both the algorithms were coded in FORTRAN 77 and tested on an IBM 4381 computer system. The test problems are generated randomly with variation in size, number of categories and density. The partition $S_1, S_2, \ldots, S_p$ is generated randomly, with the minimum number of elements fixed in each category S_i, $1 \leqslant i \leqslant p$. The elements of the cost matrix are uniformly distributed random integers in the range [1, 100]. Six problem sizes are considered and for each problem size, five different problems are solved. Results pertaining to MINMAX_GREEDY are summarized in Table 1 and that of TABU_CAS1 are summarized in Table 2. The computation time is given in seconds of CPU time. In Table 2, *iteration index* corresponds to the iteration number at which the best solution was obtained. For the results reported in Table 2 regarding TABU_CAS1, we did not use any of the diversification strategies. We simply fixed MAX_ITER = 300 and stopped the execution after this many iterations. The purpose of Table 2 is to give a feel of the running time and solution quality of TABU_CAS1 in comparison with MINMAX_GREEDY. We now consider the effects of various diversification strategies in TABU_CAS1. We have tested *fixed* tabu size with TABU_SIZE = 8, 12, 16 and 20 as well as *problem size dependent* tabu size with TABU_SIZE $\lceil n/2 \rceil$, $\lceil n/3 \rceil$ and $\lceil n/4 \rceil$ on the random problems generated in Table 2. From our computational experiments it is difficult to draw a conclusion on which TABU_SIZE strategy is better or optimum. This encourages the use of a diversification strategy with different TABU_SIZE. We also conducted similar experiments with variation in BEST_SIZE with BEST_SIZE = 1, 2, 3, 4, 5, 6 and also with and without long term memory. As in the case of TABU_SIZE we could not identify an optimum value of BEST_SIZE: this indicates the need for a diversification of search using different values for BEST_SIZE. However, we observed that it is better to keep BEST_SIZE less than or equal to 3.

TABLE 1. *Performance of MINMAX_GREEDY*

n	*p*	Objective value			Time		
		min	max	ave	min	max	ave
20	4	84	144	110.4	0.033	0.037	0.034
40	6	65	140	94.6	0.160	0.170	0.164
60	7	83	154	117.2	0.387	0.410	0.400
100	9	80	149	114.2	1.207	1.237	1.223
125	10	92	189	113.8	2.020	2.143	2.060
150	16	56	178	111.4	3.157	3.273	3.211

TABLE 2. *Performance of TABU_CAS1*

n	*p*	TABU_SIZE	Objective value			Iteration index			Time
			min	max	ave	min	max	ave	
20	4	8	42	60	51.4	18	295	95	2.13
40	6	10	36	49	42.8	31	179	90	10.06
60	7	15	40	44	41.8	89	284	178	24.71
100	9	25	41	73	49.4	1	287	151	77.68
125	10	31	45	65	53.2	3	220	116	130.58
150	16	37	30	41	36.8	33	300	163	250.36

The use of BEST_LIST was found to be useful in several situations. For example, in a problem with $n = 125$, $p = 10$, BEST_SIZE = 1, TABU_SIZE = 8, the starting greedy solution did not improve even after 400 iterations. But, when used with BEST_SIZE = 3, the algorithm produced an improved solution with objective function value one-half that of the greedy solution. This best solution was obtained at the 347th iteration. The use of long-term memory was also observed to be very effective in diversifying the search path and obtaining improved solutions.

CONCLUSION

In this paper, we have considered problems CAS1 and CAS2 and presented efficient heuristics to solve CAS1. Similar heuristics can be easily obtained for CAS2 also. The major purpose of the paper is twofold. On the one hand, we present efficient heuristics to solve the NP-hard problems CAS1 and CAS2 for which no heuristic is available in literature. On the other hand, we propose a new search diversification strategy in tabu search using BEST_LIST. BEST_LIST can be easily incorporated in any tabu search algorithm.

Although, TABU_CAS1 improves the MINMAX_GREEDY solution considerably, a major drawback of TABU_CAS1 is the $O(n^2 \log p)$ time taken per iteration. The factor n^2 is due to size of the neighbourhood of search. The search neighbour can be restricted to a part of all pair-wise interchanges by using a candidate list strategy as proposed by Glover[13] and is expected to reduce running time without sacrificing significantly the solution quality. This approach has been tested recently by Skorin-Kapov[14] for the quadratic assignment problem with good computational results. The algorithm TABU_CAS1 can possibly be further improved by using a more refined form of TABU_LIST using dynamic TABU_SIZE and 'moving gaps'.[10, 14] Integrating techniques such as 'target analysis' and 'frequency/recency based intermediate and long term memory'[10, 15] are also expected to produce good quality solutions for large size problems. Effects of new neighbourhoods, a more refined aspiration Criterion and a new kind of 'value of a move'[16] also need to be examined. Along the lines of References 17 and 18, TABU_CAS1 can also be parallelized to reduce running time substantially. Implementation of TABU_CAS1 incorporating the aforementioned ideas is an interesting avenue for future research especially when good real-life applications warrant such a study.

REFERENCES

1. R. E. BURKARD and U. DERIGS (1980) Assignment and matching problems: solution methods with Fortran programs. Lecture Notes in Economics and Mathematical Systems, No 184, Springer Verlag, Berlin.

2. C. R. Seshan (1981) Some generalisations of the time minimising assignment problem. *J. Opl Res. Soc.* **32**, 489–494.
3. V. Aggarwal, V. G. Tikekar and Lie-Fer Hsu (1986) Bottleneck assignment problem under categorisation. *Comps Opns Res.* **13**, 11–26.
4. M. B. Richey and A. P. Punnen (1989) Minimum weight perfect bipartite matching and spanning trees under categorisation. Presented at CORS/ORSA/TIMS Conference, Canada.
5. M. R. Garey and D. S. Johnson (1979) *Computers and Intractability.* Freeman, San Francisco, CA.
6. H. Kellerer and G. Woeginger (1990) A tight LPT bound for 3-partition. Report No: 170, Institute für Mathematik, Technische Universitat Graz, Austria.
7. R. L. Graham (1969) Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.* **17**, 416–429.
8. F. Glover (1989) Tabu search—Part (i). *ORSA J. Computing* **1**, 190–206.
9. F. Glover (1990) Tabu search—Part (ii). *ORSA J. Computing* **2**, 4–32.
10. F. Glover (1989) Tabu search—A tutorial. Technical Report, Graduate School of Business Administration, University of Colorado at Boulder.
11. J. Skorin-Kapov (1990) Tabu search applied to the quadratic assignment problem. *ORSA J. Computing* **2**, 33–45.
12. M. L. Fredman and R. E. Tarjan (1984) Fibonacci heaps and their uses in network optimisation algorithms. *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, pp. 338–346.
13. F. Glover (1989) Candidate list strategies and tabu search. Technical Report, Centre for Applied Artificial Intelligence, University of Colorado at Boulder.
14. J. Skorin-kapov (1990) Extensions of a tabu search adaptation to the quadratic assignment problem. Technical Report HAR-90-006, Harriman School for Management and Policy, SUNY at Stony Brook, NY 11794.
15. M. Laguna and F. Glover (1991) Integrating target analysis and tabu search for improved scheduling systems. Technical Report, Graduate School of Business Administration, University of Colorado at Boulder.
16. F.Glover and R. Hubscher (1991) Bin packing with tabu search. Technical Report, Centre for Applied Artificial Intelligence, University of Colorado, Boulder.
17. J. Chakrapani and J. Skorin-kapov (1991) Massively parallel tabu search algorithm for the quadratic assignment problem. Harriman School Working Paper HAR-91-06, SUNY at Stony Brook, NY 11794.
18. E. Taillard (1991) Robust tabu search algorithm for the quadratic assignment problem. *Parallel Computing* **17**, 443–455.