11-2003

Manpower Allocation with Time Windows and Job Teaming Constraints

Xi LI
Hong Kong University of Science and Technology

Yan Zhi LI
Hong Kong University of Science and Technology

Andrew LIM
Hong University of Science and Technology

Brian RODRIGUES
Singapore Management University, brianr@smu.edu.sg

MANPOWER ALLOCATION WITH TIME WINDOWS AND JOB TEAMING CONSTRAINTS

X. Li[1], Y.Z. Li[2], A. Lim[2] and B. Rodrigues[3]

[1]Dept of Computer Science,National University of Singapore,
3 Science Drive 2,Singapore
lixiaoc@comp.nus.edu.sg

[2]Dept of IEEM,Hong Kong University of Science and Technology,
Clear Water Bay, Hong Kong.
{ieyanzhi,iealim}@ust.hk

[3]School of Business,Singapore Management University,
469 Bukit Timah Road,Singapore 259756
br@smu.edu.sg

ABSTRACT

In the Manpower Allocation Problem with Time Windows and Job-Teaming Constraints (MAPTWTC), we have a set of jobs located at various locations where each job requires a team of workers. Each job has a time window and a job duration, during which everyone on the team has to be present. The job requirement is satisfied if and only if the required composite team works for long enough duration within the job's time window. The objective of the problem is find a schedule to minimize a weighted sum of the total number of workers, the total travelling distances of all workers and their total waiting time. Two main approaches are proposed in the paper which are shown to be able to obtain very good performance.

Keywords: Workforce planning,Heuristics,Optimization,Work teams

INTRODUCTION

In present-day business environments, groups of workers in the form of teams are common. We often see long lists of project team meetings in the busy schedules of investment bankers, mid-level business representatives or management. A team may consist of people with different expertise. They can be easily deployed, and can be formed and disbanded flexibly. Moreover, team members may arrive at a job location separately, as is commonly the case and since this allows for greater flexibility of deployment. In such situations, the efficient management of such teams in the context of manpower allocation and scheduling becomes a priority[1].

As an example, consider a media company which shoots programs at various sites. Each site may need a number of directors, coordinators, cameramen, lighting staff, recorders, technicians and stage workers. Programs can share staff members like recorders, cameramen, technicians and stage workers since their jobs are not quite unique. Each program shoot has its duration, and in particular, it may need a type of worker for just a fraction of the entire shooting duration (e.g. stage worker for stage setup). In such a setting, an efficient sharing scheme of staff members among the various programs may save the corporation staffing costs. This staffing problem is more conspicuous in organizing large-scale activities such as large conferences or sports events. Yet another potential application would be in military planning for special forces teams comprising of members with different expertise and who can arrive to form teams at different time epochs.

The rest of this paper is organized as follows. In next section we first formally state our problem. Two approaches to tackle this NP-hard problem are presented in the following two sections. And then experimental results of various approaches are compared and analyzed. In the final section we conclude our paper.

PROBLEM STATEMENT

In the Manpower Allocation Problem with Time Windows and Job-Teaming Constraints (MAPTWTC), we have a set of projects or meetings or jobs (collectively termed jobs) located at various locations where each job requires a team of workers. We can have different professions of workers in a team where the expertise make-up of the team can be given as a requirement for that job. All workers are sent from a single location called a depot. Each job has a time window that specifies the earliest time the job can begin and latest time by which the job has to be completed. Each job has a duration, during which everyone on the team has to be present (pre-emption is not allowed , for example, in a negotiation or a project team meeting). The job requirement is satisfied if and only if the required composite team works for the required duration within the job's time window. This implies that after a worker comes to the site of a job, he/she has to wait for all his/her teammates to arrive and then, possibly wait until the start time of the job. They have to be at the job's location for the entire duration of that job. We are given a map indicating the time of travel between every pair of job locations and assume that every worker travels at the same speed. The objective of the problem is to minimize a weighted sum of the total number of workers, the total travelling distances of all workers and their total waiting time. The weights represent the relative importance of these numbers to the management.

Let's formally state our problems as follows. Given m types of workers, $T_1, ..., T_m$; n jobs, job $i(i = 1, ..., n)$ is described as a three-tuple $\{[v_{i1}, ..., v_{im}], p_i, [s_i, e_i]\}$ where $[v_{i1}, ..., v_{im}]$ denotes requirements for each type of workers, p_i is the duration of this job and $[s_i, e_i]$ is its time window. All the workers start from a depot, denoted 0. Job i is located in location i and time to travel between locations i and j is $d_{ij},\ i, j = 0, 1, ...n,\ i \neq j$. It's possible that two locations have no direct link. We are asked to give a schedule to minimize the weighted sum of total number of different workers L,total travel time T and total waiting time W of all workers, which can be denotes as $\alpha L + \beta T + \gamma W, \alpha + \beta + \gamma = 1$,where α, β, γ are relative weights.

This problem is NP-hard which can be proved by reducing the well-known NP-hard problem TSP to it.

SIMULATED ANNEALING ON CONSTRUCTION HEURISTIC

Given a jobs sequence, we can form a schedule by some construction heuristics. In the first approach, we adopt a simulated annealing framework and propose two types of construction heuristics–*simple-append* and *block-insertion*. In simulated annealing, the neighborhood of a sequence is generated by *block transposition* and *block reverse* operators. We'll sketch these core components in this section.

In *simple-append*, we read in jobs one by one from the the jobs sequence and construct a schedule incrementally. For job i, we find a set from all the current workers who can arrive at location i before $(e_i - p_i)$,the latest starting time of job i. If more than enough workers are available for job i, we randomly choose what we need from the workers set. Otherwise we simply hire new necessary workers. Job i is started when all the workers have arrived. If the arriving time is earlier than s_i, we set starting time to s_i. *simple-append* has the following weakness. If in a worker's optimal schedule, job i comes before job j while i comes after

j in the given permutation, for sure the optimal schedule can't be found. *block-insertion* addresses the defect to some extent.

In *block-insertion*, for a job i just read in from the sequence, we compute an optimal subset from the scheduled workers such that: *1)* All the workers in the subset have a common time interval that is within the job's time window; *2)* The subset results in a least number of new hires for this job. The scheduled start time of job i is the start time of the common interval that makes the subset eligible. If more workers are needed, we hire new ones. To compute an optimal subset, we first search all the current workers' schedules to see whether there exists a long enough interval for the insertion of job i. We collect all the intervals into a set I and they apply the following $O(|I|^2)$ algorithm to find the optimal subset:

- Sort starting and ending points of intervals in I together into an array in non-decreasing order. Create a stack.
- Scan the array from beginning one bye one. If the scanned point is a starting point of a interval, push it to the stack.
- Otherwise, suppose it is end point e of interval v. Examine the stack from top to bottom to search for a starting point s s.t. (s, e) is a long enough interval. Evaluate this interval and update the best found. Remove starting point s from the stack and proceed to scanning procedure.

So now given a jobs permutation, we can construct a feasible schedule from it. Our simulated annealing can be viewed as search on the permutation space. We find neighbors of the current permutation by *block transposition* and *block reverse* operators. Figure 1 visualizes these two operators.We select the starting and end points of the selection region randomly. For n jobs, the whole search space has $n!$ permutations. *Block transposition* and *block reverse* can generate n^4 and n^2 neighbors respectively. So we will not jump between distant points in the search space with these two operators. In the simulated annealing

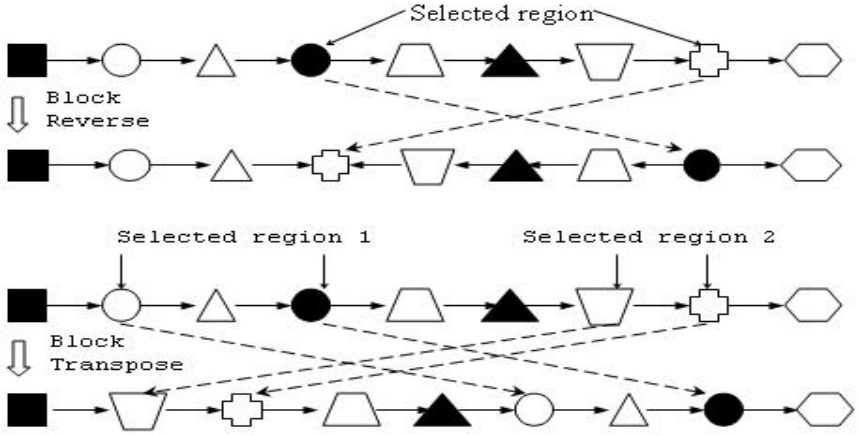

Figure 1: Neighborhood operators *Block Reverse and Block Transposition*

scheme,we use annealing with at most five reheating times. See [2] for a detained discussion of simulated annealing strategies. In the local search, the size of neighborhood and the largest distance between the nearest neighbors in the same neighborhood depends on the ratio of current temperature and the starting temperature. Higher ratio results in more neighbors in one neighborhood and more distant neighbors in the same neighborhood. The

simulate annealing on construction heuristics approach shows quite good performance in experiments.

NETWORK FLOW MODELS

Given a time scheduling of all jobs, we can show that the problem to minimize the total number of workers can be transformed to a maximum flow problem(*MFP*). *MFP* can be solved optimally and efficiently. Thus our original problem will be changed to look for the best one in the time scheduling space. In the following of this section we first illustrate the transformation to *MFP* and then show how to find the best time scheduling.

Suppose a fixed time scheduling for all jobs is given. We say job j is *compatible* to job k if a worker can serve j in specified time interval and then arrive at location k no later than the specified start time of job k. We note that this relationship is anti-symmetric. Our *Manpower Flow Graph(MFG)* consists of all jobs as nodes. There is a directed edge from j to k if and only if pair (j, k) is compatible.Workers can travel along any directed path in *MFG* and every directed path denotes a feasible worker schedule. Notice that *MFG* is the same for all types of workers. So the discussion in this section will focus on optimization of one single worker type,say, T_i.

We label the edges to represent worker flow from job to job. Edge $e = (j, k)$ is labelled with $label(e)$ if there are $label(e)$ workers of type T_i who perform job j and then travel to location k immediately. It is easy to see that the *MFT* is feasible if, for any job j, the following two upper bounds hold:

$$\sum\nolimits_{\forall k,\ e=(k,j)\in MFG} label(e) \leq v_{ij} \tag{1}$$

$$\sum\nolimits_{\forall k,\ e=(j,k)\in MFG} label(e) \leq v_{ij} \tag{2}$$

The first bound says that we require no more than v_{ij} workers type T_i to come to j and the second bound says that the number of T_i workers who depart from j can not be larger than v_{ij}.

If we are given a labelled *MFG*, it's easy to recover schedules of type T_i workers. Since for a given fixed time schedule no cycle exists in the graph, we just need travel all the paths with nonzero labels to get all the schedules. If there exist unsatisfied job requirements, we simply hire extra workers. We note that in such a sitting, the number of extra type T_i workers needed for job j equals: $v_{ij} - \sum_{\forall k,\ e=(k,j)\in MFG} label(e)$. Summing up, for all jobs, the total number of T_i workers needed for the given time schedule is: $\sum_{j=1}^{n} v_{ij} - \sum_{e\in MFG} label(e)$. Because $\sum_{j=1}^{n} v_{ij}$ is a fixed term, to get the minimum number of type T_i workers, we shall label *MFG* to maximize $\sum_{e\in MFG} label(e)$.

In order to apply maximum flow algorithms directly, we need to make a transformation for our network flow model. We spit each node j to two nodes $in(j)$ and $out(j)$. For each edge (j, k), add an edge $(in(j), out(k))$ with infinite capacity to the new graph,denoted as $\overline{MFG}$. Source s and sink t are added to $\overline{MFG}$ and for each node j,add two edges $(s, in(j))$, $(out(j), t)$ both with capacity v_{ij}.

There is one-to-one correspondence between edges in *MFT* and edges in $\overline{MFG}$ which are incident neither on s nor on t. Thus,

$$\sum\nolimits_{e\in MFG} label(e) = \sum\nolimits_{\forall k, e\neq(k,t)\wedge e\neq(s,k)}^{e\in\overline{MFG}} flow(e) = flow(\overline{MFG}) \tag{3}$$

The first equality is established by correspondence between edges in the two graphs and the second equality can be easily seen as due to the mass balance constraint on the max-flow problem.

We claim that any feasible flow in $\overline{MFG}$ corresponds to a feasible labelling in MFG because incoming edges to sink node enforce MFG upper bound (1) and outgoing edges from source node enforce MFG upper bound (2). On the other hand, the mass balance constraint in $\overline{MFG}$ does not add any more restrictions to flow in MFG because for any nodes j, k and any feasible label of $(*, k)$ and $(j, *)$ in MFG, upper bounds (1) and (2) dictate that the edges $(s, in(k))$ and $(out(j), t)$ have sufficient capacity to balance the nodes $in(k)$ and $out(j)$. Thus, any feasible labelling in MFG has a corresponding balanced flow in $\overline{MFG}$. In the light of that, a maximum flow in $\overline{MFG}$ corresponds to a maximum labelling in MFG.

Meta-heuristic searches for optimal job time schedules

We have shown that given a fixed scheduling, exact optimal total number of workers can be obtained. Since usually total number of workers is our most important aim, we can change our problem to find the best time scheduling.

Two meta-heuristic approach are implemented. One is *Ant Colony Optimization(ACO)* and the other is *Tabu embedded Simulated Annealing(TSA)*. ACO is a meta-heuristic method that simulates the process of ants as they seek for food[3]. We have implemented a standard ACO approach while it shows less powerful than TSA approach.

The simulate annealing scheme of TSA is the same as presented in the solution used on the construction heuristics. However, when we generate neighbors in TSA we first check against tabu list and only investigate those neighbors that are not tabu(see [4] for an overview of tabu search). A solution is represented by a set of time slot choices. The data is put in an array,$choice[1..n]$,with $choice[j]$ denoting the choice of time slot for job j. To generate the neighborhood, for each solution $choice[1..n]$, the algorithm randomly selects a time slot within a range of the current solution. The higher the temperature is, the larger the range size. For each possible choice of a specific job, the tabu list keeps a record on the times this choice has been chosen recently. If the times exceeds a limit, this choice is tabued. When making a neighborhood move on the current solution, we calculate how many of its choices are tabued. If the number of tabued choices exceeds a certain preset ratio measure relative to the total number of jobs, the neighbor is tabued. When too many choices are tabued, we will have a lot of neighbor moves rejected by the tabu data structure;if so,the algorithm will forget part of the tabu list.

EXPERIMENTAL RESULTS

Test cases generation

Since no benchmark data is available for the new MAPTWTC, we use newly generated test cases. In each test case, we specify a set of clusters. Each cluster is a circle with a given central point and a radius. The number of jobs in a unit area is constant in each cluster. Each cluster specifies the number of jobs in it, a time distribution of jobs and a worker type demand distribution. A important parameter is the ratio between job duration and job time window, which influences the performance of our lower bound.

Lower bounds

The lower bounds of number of hires are calculated using a relaxation of our network flow model. There is one major relaxation in *Relaxed Manpower Flow Graph*($RMFG$)

compared with MFG. The definition of *compatible* relationship is relaxed. Without given a fixed time scheduling,if a worker is *possible* to perform job k immediately after job j, we say job j is *compatible* to job k. We can see any feasible MFG must be a sub-graph of the $RMFG$; thus $RMFG$ must have a maximal flow no less than any MFG does. So we obtain a lower bound by solving maximum flow on the $RMFG$. The bound is loose if the $RMFG$ contains many cycles. A low ratio of job duration to time window leads to many cycles and thus loose bounds.

Results analysis

We have implemented different approaches presented above and extensive experiments have been conducted. Due to space limitation, we may not give details of the experimental results. Here are conclusions drawn from the results.

- In the scenarios of high ratio(≥ 0.8)–job duration to time window, all our algorithms achieve solutions less than 10% over lower bounds. Since the lower bound is possibly not tight, such performance is satisfactory. Among all the approaches, the simulated annealing scheme on block-insertion heuristic(SABI) method achieves best performance.

- In scenarios of lower ratio of job duration to time window, results of our algorithms have considerable gaps to lower bounds. This is expected since many cycles exists in $RMFG$ for lower ratios and thus we may often get infeasible solutions when seeking for a lower bound. SABI approach also performs the best in this situation.

CONCLUSIONS

In this paper, we studied the manpower allocation problem with time window and teaming constraints. This is a new and relevant problem in manpower scheduling. Two approaches were proposed to tackle this NP-hard problem. One is simulated annealing on construction heuristics and the other utilizes network flow model and embeds it in the simulated annealing framework as well. The experiments showed our algorithms performed well.

Future work may be done on tight lower bounds and more complicated MAPTWTC such as considering different costs for hiring different types of workers.

References

[1] Abboud, N., Inuiguchi, M., Sakawa, M., Uemura, Y., (1998), *Manpower allocation using genetic annealing*, European Journal of Operational Research, 111:405-420

[2] Dowsland K. (1993) Simulated annealing. In Reeves C. R. (1995) Modern Heuristic Techniques for Combinatorial Problems. Blackwell Scientific Publications, Oxford

[3] Garbardella, L.M. (2000), *An Ant Colony System Hybridized with A New Local Search for the Sequential Ordering Problem*, Informs Journal on Computing 3, pp. 237 - 255.

[4] Glover, F., Laguna, M., (1997), Tabu Search, Kluwer Academic Publishers