MAXIMIZING THE TOTAL NUMBER OF ON TIME JOBS ON IDENTICAL MACHINES

Hairong Zhao

Department of Mathematics, Computer Science & Statistics
Purdue University, Northwest

ABSTRACT

This paper studies the job-scheduling problem on m ≥ 2 parallel/identical machines.There are n jobs, denoted by J_i,,1 ≤ i ≤ n. Each job J_i, has a due date d_i. A job has one or more tasks, each with a specific processing time. The tasks can't be preempted, i.e., once scheduled, a task cannot be interrupted and resumed later. Different tasks of the same job can be scheduled concurrently on different machines. A job is on time if all of its tasks finish before its due date; otherwise, it is tardy. A schedule of the jobs specifies which task is scheduled on which machine at what time. The problem is to find a schedule of these jobs so that the number of on time jobs is maximized; or equivalently, the number of tardy jobs is minimized. We consider two cases: the case when each job has only a single task and the case where a job can have one or more tasks. For the first case, if all jobs have common due date we design a simple algorithm and show that the algorithm can generate a schedule whose number of on time jobs is at most (m-1) less than that of the optimal schedule. We also show that the modified algorithm works for the second case with common due date and has same performance. Finally, we design an algorithm when jobs have different due dates for the second case. We conduct computation experiment and show that the algorithm has very good performance.

KEYWORDS

On time job, identical machines, order scheduling

1. INTRODUCTION

This paper studies the job-scheduling problem on m ≥ 2 parallel/identical machines. There are n jobs, denoted by $J_{i,,}$1 ≤ i ≤ n. Each job J_i, has a due date d_i. A job has one or more tasks, each with a specific processing time. The tasks can't be preempted, i.e., once scheduled, a task cannot be interrupted and resume later. Different tasks of the same job can be scheduled concurrently on different machines. A job is on time if all of its tasks finish before its due date; otherwise, it is tardy. A schedule of the jobs specifies which task is processed on which machine at what time. The problem is to find a schedule of these jobs so that the number of on time jobs is maximized; or equivalently, the number of tardy jobs is minimized. The number of on time/tardy jobs is a very important criterion since, in many cases, the cost penalty incurred by a tardy job does not depend on how late it is, but the fact that it is late. In such cases, an appropriate objective would be to minimize the number of tardy jobs. For example, a late job may cause a customer to switch to another supplier, especially in the just-in-time production environment.

This problem has been studied in many literatures. In the classic model each job has a single task. When there is a single machine, i.e., m=1, Moore ([2]) gave an O(n log n) algorithm (sometimes known as Hudgson's Algorithm) that solves the problem optimally. When $m \geq 2$, the problem becomes NP hard even if all jobs have the same due date. When m = 2, Leung & Yu [1] gave a heuristic, based on Moore's algorithm, for the multiprocessor case. It is shown that the performance ratio of the heuristic is 4/3 for two identical processors, where the performance ratio is defined to be the least upper bound of the ratio of the number of on-time jobs in an optimal schedule versus that in the schedule generated by the heuristic. Ho and Chang [3] conducted extensive simulation experiment to test the effectiveness of the heuristic on multiple machines. The simulation results showed that this heuristic is quite effective in most cases. The paper also proposed two other heuristics.

The scheduling model where a job can have multiple tasks is called order scheduling in literature (see [4] and the references therein). In general order scheduling, a machine may be dedicated and can only process one type of tasks; or be flexible and can process multiple types of tasks. In this paper, all machines are identical and fully flexible, i.e. every machine is able to process all types of tasks and different tasks of an order/job can be processed concurrently. If there is one machine, the problem minimizing the number of tardy jobs under this model is reduced to the classics model where each job has only a single task. If there are multiple machines, however, the classical model is a special case of order scheduling. Some work has been done for this model with the objective of total completion time. When the jobs are unweighted, Blocher and Chhajed ([5]) show that the problem is ordinary NP-hard for any fixed $m \geq 2$ and strongly NP-hard when m is arbitrary. Then the authors presented six heuristics and performed empirical analysis of the heuristics. Two classes of nine heuristics with proven worst-case performance bounds were studied by Leung, L_i and Pinedo in [6]. To the best of our knowledge, no past work has ever been done for the number of on time jobs objective.

In this paper, we are interested in the performance of simple heuristics for both the classical model and the order-scheduling model. We first consider the case with common due date, then we consider the general case where jobs can have arbitrary due date. For the general case, we evaluate the performance of the heuristics by some experimental results.

Note that for the optimal solution, the problem of minimizing the number of tardy jobs is the same as maximizing the number of on time jobs. However, all our problems are NP-hard so there is no hope to find an optimal schedule in polynomial time. We can only design effective and efficient heuristics for large size problems. To evaluate the effectiveness of a heuristic, we could use the absolute error, i.e. the difference between the optimal solution and the solution found by the heuristic; we may also use relative error, i.e the ratio of the absolute error and the optimal solution. In this case, if we use the number of tardy jobs, it is possible that the optimal schedule has 0 tardy jobs, and consequently the relative error becomes infinity. It is for this reason; we consider the number of on time jobs, instead of tardy jobs in this paper.

2. Classical Model With Common Due Date

In this section, we assume that each job has a single task and all jobs have the same common due date. Using the three field notation, the problem of minimizing the number of tardy jobs can be denoted as $P_m|d_j=d\ |\Sigma U_j$ where $U_j=1$ if J_j is tardy.

Algorithm1:

Input:

- J= { J_1, J_2 …, J_n}, a set of n jobs, job Ji in has processing time pi and due date d
- m machines, M_1, M_2, …, M_m

Output:

A non-preemptive schedule of a subset of J where all jobs are on time.

Method: Schedule the jobs in SPT order (Shortest Processing Time first) on M1 before d, if no more jobs can be scheduled on M1, and then schedule the jobs on M2 before d, and keep repeating this procedure until all jobs have been scheduled or no more jobs can be scheduled on M_m before d.

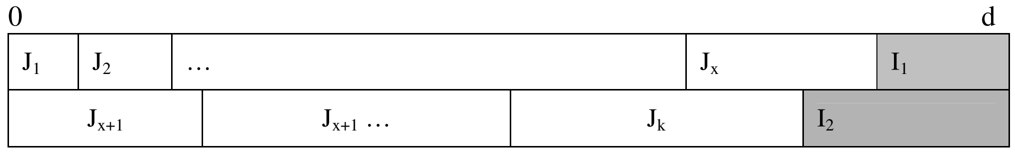
Algorithm1 certainly is not optimal. Consider the example of four jobs and two machines such that the processing times are, 1, 2, 3, 4 and the common due date 5. Algorithm1 will schedule the first two jobs on M_1, and the third job on M_2, and the last job is tardy. However, the optimal schedule will schedule the first job and the last job on M_1, and the second and the third job on M_2. The absolute error is one. It turns out this is not a coincident. We have the following Lemma.

Theorem1. For two machines, the number of on time jobs of the schedule that is generated byAlgorithm1 is at most one less than that of the optimal schedule, i.e. the absolute error of is at most 1.

Proof. For convenience, suppose the jobs are numbered in the SPT order. Suppose there are k on time jobs in the schedule generated by Algorithm1, then it must be the first k shorted jobs. Suppose that the schedule is not optimal. Let I_1 and I_2 be the length of idle interval on M1 and M2, respectively. Then we must have that $I_1 < p_{k+1}$ and $I_2 < p_{k+1}$. Thus, the total processing time of on time jobs before d in any schedule is at most

$$p_1 + p_2 + .. + p_k + I_1 + I_2 < p_1 + p_2 + .. + p_k + 2\, p_{k+1.}$$

Since we schedule the jobs in SPT order, in any other schedule, at most k+1 jobs can be scheduled before d.



Using similar argument, we can prove the following theorem.

Theorem2. For m machines, the number of on time jobs of the schedule that is generated by Algorithm1 is at most (m-1) less than that of the optimal schedule, i.e. the absolute error of is at most (m-1).

3. ORDER SCHEDULING MODEL

In this section, we consider the case that a job may have one or more tasks. For job J_i, we use, $J_{i,1,}$ $J_{i,2,}$ … to represent its tasks, and we use $p_{i,1}$, $p_{i,2}$, … to represent the length of these tasks. We still use pi to represent the total length of the job. Different tasks of the same job can be scheduled concurrently on different machines. We consider 2 cases, the jobs have common due date and the case each job has their own due date. Without loss of generality, we assume that for each job, $p_j \leq d_j$.

3.1. Common Due Date

Modified Algorithm1: We still consider the jobs one by one in SPT order, and for each job, schedule the tasks in arbitrary order; for a particular task, first check if it can be scheduled on the first machine before the due date, if not, then check if it can be scheduled on the second machine, and so on. If the task can't be scheduled to any machine before the due date, then this job and remaining jobs will be the tardy jobs. Otherwise, we repeat this procedure until all jobsare scheduled before the due date or until no more jobs can be scheduled on time.

Using similar argument as the proof of theorem1, we can prove the following theorem.Theorem3. For m machines, the number of on time jobs of the schedule that is generated by Modified Algorithm1 is at most (m-1) less than that of the optimal schedule.

3.2. Different Due Date

Algorithm2: Consider the jobs EDD (Earliest Due Date First) order. For each job, schedule the tasks one by one in arbitrary order; to schedule a task $J_{i,x}$, choose the machine with the latest finishing time such that $J_{i,x}$ can finish before its due date d_j. If there is no such machine, it means J_i or one of the jobs that have been scheduled on time has to be tardy.

To find out which one is a better choice for the tardy job, we do the following:

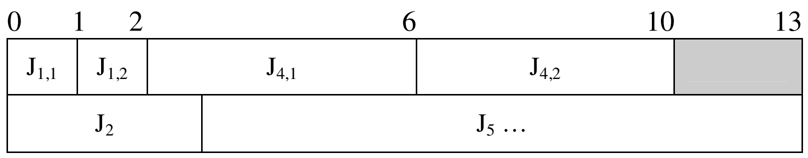
- Restore the schedule by deleting those tasks of J_i that have been scheduled. Let S be the schedule, and f_{max} be the maximum finishing time of the jobs that have been scheduled in S.
- Find the job J_k with the largest processing time from those jobs that have been scheduled.
- Deleting job J_k from the schedule, try to schedule job J_i.
- If some of the tasks of J_i can't be scheduled on time, Ji will be chosen as a tardy job. Otherwise, let S' be schedule obtained, and let f'_{max} be the maximum finishing time of the tasks of J_i in S'. If $f_{max} \leq f'_{max}$, J_i will also be chosen as the tardy job, else J_k will be chosen as the tardy job.

After we choose J_k or J_i as the tardy job, continue to schedule the remaining jobs to S' if J_k is tardy or S if J_i is on time. We terminate the process until all jobs have been scheduled on time or chosen as a tardy job.

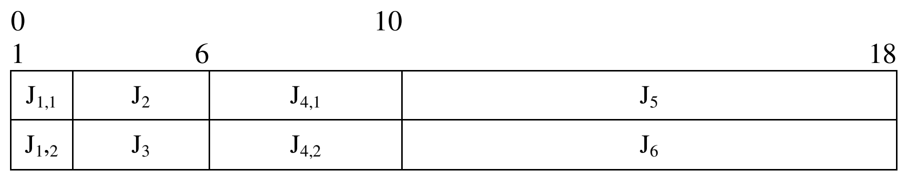
Following is an example of applying Algoirhtm2. Suppose there are 2 machines, 6 jobs.

J_1 has 2 tasks of length 1, 1, and due date 4. J_2 and J_3 have both single task of length 3, and due date 4. J_4 has 2 tasks of length 4, 4, and due date 10. J_5 and J_6 have both single task of length 10, and due date 18.

Algorithm 2 schedules J_1 J_2, J_4 and J_5 before their due dates, and two other jobs are tardy, as in the following figure.



In the optimal schedule, however, all 6 jobs are on time.



One can generalize the above example to more jobs, so that the optimal schedule has all jobs on time, but the schedule generated by Algorithm 2 has 2/3 of the jobs on time. Thus the absolution error can be quite large.

4. COMPUTATIONAL RESULTS

In this section, we want to design experiments to find the performance of Algorithm2 for randomly generated large instances.

We choose the number of jobs n = 500, and the number of machines m = 20. Problem instances of varying hardness are generated according to different characteristics of the due dates. First of all, for each job, the number of tasks is randomly generated from the uniform distribution [1,10m]. Then, the length of each task is generated from the uniform distribution [1,100]. Finally, after all jobs are generated, for each job, its due date is generated from the following uniform distribution: P/m[(1- δ2 - δ1/2), (1- δ2 + δ1/2)] where P is the total processing time of all the tasks, δ1 and δ2 determines the range in which the due dates lie and adjusts the tightness of the due dates, respectively. We set δ1 = 0.2, 0.4, 0.6, 0.8, 1.0 and δ2 = 0.6, 0.8, 1.0. For each combination of δ1 and δ2, 100 instances are generated. Thus, there are 2500 instances in total. The algorithms are implemented in Java.

To evaluate the algorithm, for each generated instance Ii (i = 1, 2, . . . , 100), we construct the corresponding single-machine instance I'_i as described follows:

For each job j, construct a job j for I'i with processing time $p'_j = p_j/m$ and due date d′ = d.
The instance I'_i can be solved optimally by Moore's algorithm, and one can show that the number of on time jobs of I'_i, denoted by $UB(I'_i)$, is an upper bound of the number of on time jobs of the original instance I_i. We compare the number of on time jobs of the schedule found by Algorithm2 with its corresponding upper bound.

Our result shows that the absolute error in most cases is less than one, i.e. Algorithm 2 finds optimal schedule in most cases. We also calculated the average relative error with respect to the upper bound, and the result is summarized as follows.

(δ1, δ2)	(0.2,0.2)	(0.2,0.4)	(0.2,0.6)	(0.2,0.8)	(0.2,1.0)
relative error	0.10%	0.10%	0.20%	0.20%	0.10%
(δ1, δ2)	(0.4,0.2)	(0.4,0.4)	(0.4,0.6)	(0.4,0.8)	(0.4,1.0)
relative error	0.10%	0.10%	0.10%	0.20%	0.10%
(δ1, δ2)	(0.6,0.2)	(0.6,0.4)	(0.6,0.6)	(0.6,0.8)	(0.6,1.0)
relative error	0%	0.10%	0.10%	0.10%	0.10%
(δ1, δ2)	(0.8,0.2)	(0.8,0.4)	(0.8,0.6)	(0.8,0.8)	(0.8,1.0)
relative error	0.00%	0.10%	0.10%	0.10%	0.10%
(δ1, δ2)	(1.0,0.2)	(1.0,0.4)	(1.0,0.6)	(1.0,0.8)	(1.0,1.0)
relative error	0%	0%	0.10%	0.30%	0.60%

5. CONCLUSIONS

This paper studies the problem of scheduling n jobs to m identical machines with the objective of maximizing the number of on time jobs. Each job J_i, has a due date d_i. A job has one or more tasks and the tasks can't be preempted, but different tasks of the same job can be scheduled concurrently on different machines. We considered two cases: the case when each job has only a single task and the case where a job can have one or more tasks. We designed a simple and effective algorithm when all jobs have common due date. We also designed an algorithm when jobs have different due dates for the second case. We conducted computation experiment and showed that the algorithm has very good performance.

REFERENCES

[1] J Y-T, Leung and V. K.M. Yu, Heuristic for minimizing the number of late jobs on two processors, International Journal of Foundations of Computer Science, Vol 5, Nos 3 & 4 (1994), pp. 261-279.

[2] J. M. Moore, An n job, one machine sequencing algorithm for minimizing the number of late jobs, Management Science, 15 (1968), 102–109.

[3] J. C. Ho and Y. L. Chang, Minimizing the number of tardy jobs for m parallel machines, European Journal of Operational Research, 8(2), 1995, 343-355.

[4] J.-T. Leung, H. Li, M. Pinedo, Order scheduling models: an overview, in: G. Kendall, E. K. Burke, S. Petrovic, M. Gendreau (Eds.), Multidisciplinary Scheduling: Theory and Applications, Springer, 2005, pp. 37–53.

[5] J. Blocher, D. Chhajed, The customer order lead-time problem on parallel machines, Naval Research Logistics 43 (1996) 629-654.

[6] J.-T. Leung, H. Li, M. Pinedo, Approximation algorithms for minimizing total weighted completion time of orders on identical machines in parallel, Naval Research Logistics 53~(4) (2006) 243--260.