Advances in Engineering Research (AER), volume 130
5th International Conference on Frontiers of Manufacturing Science and Measuring Technology (FMSMT 2017)

A Simple Algorithm for Scheduling Jobs with GoS Levels

Xiao Xin
College of Foreign Studies, Shandong Institute of Business and Technology, Yantai, 264005, China
xinxiaoyt@hotmail.com

Keywords: Scheduling, Grade of Service, Makespan, Worst-case Analysis.

Abstract. The problem of scheduling jobs with grade of service (GoS) levels is considered, which is motivated from an application in service industry. Jobs and machines are labeled with GoS levels. If the GoS level of a job is greater than or equal to the GoS level of a machine, then this job can be processed on this machine. The objective is to minimize makespan. A simple and fast 3/2-approximation algorithm is presented.

Introduction

In service industry, special customers (such as silver, gold, or platinum members) are usually entitled to premium services by the service providers. Providing differentiated services can be modeled as the following scheduling problem. There are machines and jobs which are labeled with grade of service (GoS) levels. Jobs correspond to the service requests from the customers. Relatively higher levels are labeled on the jobs from valued members. Machines correspond to the services provided by the service providers. Relatively higher levels are labeled on the machines corresponding to the premium services. If the GoS level of a job is not less than the GoS level of a machine, then this job can be processed on this machine. In effect, the valued members are ensured to receive better services than the regular members.

For more formal definition of the problem, let $\mathcal{J}=\{1,2,\ldots,n\}$ and $\mathcal{M}=\{M_1,M_2,\ldots,M_m\}$ denote the sets of jobs and machines, respectively. Job j ($j=1,2\ldots,n$) has a *processing time* p_j and is labeled with a *GoS level* g_j. Machine M_i ($i=1,2\ldots,m$) has a GoS level $g(M_i)$. If $g_j \geq g(M_i)$, then machine M_i is allowed to process job j and is called an *eligible machine* for job j. A *schedule* is a partition of $\mathcal{J}$ into m disjoint subsets $S_1,S_2,\ldots S_m$, where S_i denotes the set of the jobs assigned to machine M_i. The jobs must be assigned validly to the machines. That is, $j \in S_i$ only if $g_j \geq g(M_i)$, $i=1,2\ldots,m$, $j=1,2\ldots,n$. The *load* of machine M_i in this schedule, denoted as $Load_i$, is defined to be $\sum_{j\in S_i} p_j$, i.e., the total processing time of the jobs processed on M_i, $i=1,2\ldots,m$. The *makespan* of this schedule, denoted as $C_{\max}$, is defined to be $\max_{1\leq i\leq m} Load_i$, i.e., the maximum load of all machines. The goal is to find a schedule such that $C_{\max}$ is minimized. Following the notational convention developed by Graham et al. [1], we denote this problem as $P|GoS|C_{\max}$. It is called the problem of *scheduling with GoS levels* [2].

Scheduling with GoS levels is a special case of the following *scheduling with processing set restrictions*. Each job j has to be processed by a machine that belongs to a specific subset $\mathcal{M}_j$ of the m machines called its *eligible set*. See [3] for a recent survey on machine scheduling with processing set restrictions. The survey covers five types of processing set restrictions, namely *inclusive*, *nested*, *interval*, *tree-hierarchical*, and *arbitrary*. Scheduling with GoS levels is equivalent to scheduling with inclusive processing set restrictions: for any two jobs j_1 and j_2, either $\mathcal{M}_{j_1}\subseteq\mathcal{M}_{j_2}$ or $\mathcal{M}_{j_2}\subseteq\mathcal{M}_{j_1}$.

If the smallest GoS level of the jobs is not less than the largest GoS level of the machines, then all jobs can be assigned to any machine. Hence, in this special case problem $P|GoS|C_{\max}$

becomes the classical scheduling problem $P||C_{\max}$. Likewise, if all the jobs have the same GoS level, say g, we can ignore the machines with GoS levels greater than g and thus get an instance of $P||C_{\max}$. Since $P||C_{\max}$ is strongly NP-hard [4], $P|GoS|C_{\max}$ is also strongly NP-hard.

To tackle NP-hard problems, theoretical research has been focused on designing approximation algorithms. The readers are referred to [5] for the standard definitions, such as approximation ratio, polynomial time approximation scheme (PTAS), fully polynomial time approximation scheme (FPTAS), etc.

For $P||C_{\max}$, the *list scheduling* (LS) algorithm [6], the *longest processing time first* (LPT) rule [7] and MULTIFIT [8] yield schedules with makespan no more than $2-1/m$, $4/3-1/(3m)$ and 13/11 times the optimum, respectively. It also admits a PTAS when m is part of the input [9] and an FPTAS when m is a fixed number [10].

Lenstra et al. proposed a 2-approximation algorithm for the classical unrelated machines scheduling problem $R||C_{\max}$ [11]. Since $P|GoS|C_{\max}$ is a special case of $R||C_{\max}$, the 2-approximation algorithm developed for $R||C_{\max}$ is also applicable to $P|GoS|C_{\max}$. However, the algorithm has to employ the linear programming solution module whose computational complexity is not yet proved to be strongly polynomial.

For $P|GoS|C_{\max}$, Kafura and Shen [12] proposed a $(2-1/(m-1))$ -approximation algorithm with running time $O(n\log n+(n+m)\log m)$. Azar et al. [13] presented a $(\lceil \log_2 m \rceil+1)$ -approximation algorithm which is online in nature, but its bound is an increasing function of m. Later, Hwang et al. [2] independently presented an algorithm which is the same as that presented in [12]. Glass and Keller [14] designed a 3/2-approximation algorithm with running time $O(nm\log m)$. Ou, Leung and Li [15] gave a 4/3- approximation algorithm with running time $O((n+m)(\log nm)\log p_{sum})$, a $(4/3+\varepsilon)$ -approximation algorithm with running time $O((n+m)(\log nm)\log(1/\varepsilon))$ and a PTAS with running time $O(mn^{2+(8/\varepsilon)\log(4/\varepsilon)}\log p_{sum}+m\log m)$, where $p_{sum}=\sum_{j=1}^{n}p_j$ and $\varepsilon>0$ can be made arbitrarily small.

If each job j has a release date r_j before which it cannot be scheduled, then problem $P|GoS|C_{\max}$ is extended to $P|r_j,GoS|C_{\max}$. Li and Wang [16] generalized the PTAS in [15] and got a PTAS for $P|r_j,GoS|C_{\max}$. When all jobs have equal processing times, $P|r_j,GoS|C_{\max}$ becomes $P|r_j,p_j=p,GoS|C_{\max}$. Li and Li [17] developed an exact algorithm for $P|r_j,p_j=p,GoS|C_{\max}$ with running time $O(n^2+mn\log n)$. Later, Li and Lee [18] presented another exact algorithm for $P|r_j,p_j=p,GoS|C_{\max}$ with improved running time $O(\min\{m,\log n\}n\log n)$.

The $(2-1/(m-1))$ -approximation algorithm for $P|GoS|C_{\max}$ presented in [12] works as follows. The jobs are re-indexed as $1,2,\ldots,n$ such that either (i) $g_j<g_{j+1}$, or (ii) $g_j=g_{j+1}$ and $p_j\geq p_{j+1}$. The algorithm assigns the jobs $1,2,\ldots,n$ one by one each to an eligible machine with the currently least load. The 3/2-approximation algorithm for $P|GoS|C_{\max}$ presented in [14] works as follows. First, the longest m jobs are selected. Then, for $l=0,1,\ldots,m$, a subroutine $\mathcal{B}(l)$ is invoked which attempts to assign the longest l jobs to l different machines. At this stage, each job is assigned to an empty eligible machine with the highest index. If $\mathcal{B}(l)$ succeeds, it calculates a bound $C(l)$. The remaining jobs are then scheduled. Each is assigned to an eligible machine with the highest index whose total load (plus the processing time of this job) does not exceed $3C(l)/2$. Finally, among all feasible schedules, the algorithm outputs the one with the minimum makespan.

In this paper, we combine the techniques used in [2] and [14] to get a 3/2-approximation algorithm with running time $O(nm\log m)$ for $P|GoS|C_{\max}$. The algorithm has a better approximation ratio than

the one in [12]. Although it has the same time complexity as the one in [14], it is simpler and easier to implement, since it avoids the involved computation of $C(l)$.

The remainder of this paper is organized as follows. In Section 2, we state the preliminaries. In Section 3, we present an algorithm for $P|GoS|C_{\max}$. In Section 4, we prove the correctness of the algorithm and analyze its approximation ratio. We conclude this paper in Section 5.

Preliminaries

For simplicity, we assume without loss of generality that the machines are indexed in non-decreasing order of their GoS levels such that $g(M_1) \le g(M_2) \le \cdots \le g(M_m)$. Let $g(M_{m+1}) = +\infty$. Let $\mathcal{J}_i = \{j \in \mathcal{J} \mid g(M_i) \le g_j < g(M_{i+1})\}$, $i = 1,2,\ldots,m$. Certainly, we have $\mathcal{J} = \bigcup_{i=1}^{m} \mathcal{J}_i$. The jobs in $\mathcal{J}_i$ ($i = 1,2,\ldots,m$) can be processed on any of the machines $M_1, M_2, \ldots, M_i$, but cannot be processed on any of the machines $M_{i+1}, M_{i+2}, \ldots, M_m$.

Let OPT denote the makespan of an optimal schedule for $P|GoS|C_{\max}$. The jobs are classified as long jobs and short jobs. Job j is called a *long job* if $p_j > OPT/2$; otherwise, it is called a *short job*. Let n_L denote the number of long jobs in $\mathcal{J}$. Certainly, since the exact value of OPT is not known, the exact value of n_L is not known, neither. Fortunately, since each machine can process at most one long job in any optimal schedule, we have $n_L \le m$. This allows us to perform an efficient enumeration.

The Algorithm

Let l be an estimate value of n_L. The algorithm presented in this section, called LFSN-LS (Long First Short Next-List Scheduling), follows the framework of the one in [14]. Initially, it selects the longest m jobs from $\mathcal{J}$. The selection can be done in $O(nm)$ time. Then, it tries each possible value l in the interval $[0,m]$. For each selected value l ($l = 0,1,\ldots,m$), it invokes a subroutine $\mathcal{B}(l)$ which works as follows. Subroutine $\mathcal{B}(l)$ attempts to assign the longest l jobs to l different machines. Each selected job is assigned to an empty eligible machine with the highest index. If $\mathcal{B}(l)$ fails (i.e., some selected job cannot be assigned to an empty eligible machine), then it must be true that $n_L < l$, and the enumeration completes (the values larger than the currently selected value of l will not be examined.) If $\mathcal{B}(l)$ succeeds at this moment, then for $i = 1,2,\ldots,m$, $\mathcal{B}(l)$ schedules the remaining jobs in $\mathcal{J}_i$ one by one by the *list scheduling* algorithm [6]: each is assigned to an eligible machine whose current load is the minimum among all the eligible machines of the job. The way that $\mathcal{B}(l)$ assigns these jobs is similar to the algorithm presented in [2]. Finally, among all feasible schedules, the algorithm outputs the one with the minimum makespan.

The Analysis

We now prove the correctness of the algorithm and analyze its approximation ratio.

Theorem 1. *LFSN-LS is a 3/2-approximation algorithm for* $P|GoS|C_{\max}$ *that runs in* $O(nm\log m)$ *time.*

Proof. Assume that $n_L = l^*$. Subroutine $\mathcal{B}(l^*)$ assigns each long job to an empty eligible machine (this fact has been proved in [14]), and all the other jobs are short jobs which are scheduled in non-decreasing order of their GoS levels by the list scheduling algorithm. Hence, $\mathcal{B}(l^*)$ assigns all jobs to the machines.

Let Σ denote the schedule generated by $\mathcal{B}(l^*)$ with makespan $C_{\max}$. Let j denote the first job in Σ which is completed on machine M_i at time $C_{\max}$. If j is a long job, then it is the only job processed on M_i and Σ is an optimal schedule; in this case we have $C_{\max} = OPT$. If j is a short job, let t denote the start time of job j in Σ. When job j is assigned to M_i, M_i is the least loaded machine among machines $M_1, M_2, \ldots, M_{g_j}$. If $t > OPT$, then the total processing time of the jobs assigned to machines $M_1, M_2, \ldots, M_{g_j}$ before job j exceeds $g_j \cdot OPT$. Since all these jobs have to be assigned to machines $M_1, M_2, \ldots, M_{g_j}$ in any feasible schedule, they cannot be completed by time OPT in any feasible schedule. Therefore, we have $t \le OPT$. It follows that $C_{\max} = t + p_j < OPT + OPT/2 = 3OPT/2$. Hence, Σ is a feasible schedule whose makespan is at most $3OPT/2$.

By selecting the best schedule from among the generate feasible schedules, LFSN-LS returns a feasible schedule whose makespan is at most $3OPT/2$.

For each selected value l ($l = 0,1,\ldots,m$), subroutine $\mathcal{B}(l)$ can be implemented in time $O(n \log m)$, since it takes $O(\log m)$ time to select the least loaded eligible machine for each short job. Hence, LFSN-LS is implementable in $O(nm \log m)$ time.

Conclusion

In this paper, we tried to assign job, having different processing times and GoS levels, onto parallel machines with different GoS levels. The objective is to minimize makespan. We presented a simple and fast 3/2-approximation algorithm for this problem. It would be interesting to design simple and fast algorithms with better approximation ratios, using another heuristic instead of the List Scheduling rule as a base method.

References

[1] R. L. Graham, E. L. Lawler, J. K. Lenstra and A. R. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of discrete mathematics. 5 (1979) 287-326.

[2] H.-C. Hwang, S. Y. Chang and K. Lee, Parallel machine scheduling under a grade of service provision, Computers & Operations Research. 31 (2004) 2055-2061.

[3] J. Y.-T. Leung, C.-L. Li, Scheduling with processing set restrictions: A literature update, International Journal of Production Economics. 175 (2016) 1-11.

[4] M. R. Garey, D. S. Johnson, Computers and intractability: a guide to the theory of NP-completeness, Freeman, New York, 1979.

[5] C. H. Papadimitriou, K. Steiglitz, Combinatorial optimization: algorithms and complexity, Courier Dover Publications, 1998.

[6] R. L. Graham, Bounds for certain multiprocessing anomalies, Bell System Technical Journal. 45 (1966) 1563-1581.

[7] R. L. Graham, Bounds on multiprocessing timing anomalies, SIAM Journal on Applied Mathematics. 17 (1969) 416-429.

[8] M. Yue, On the exact upper bound for the multifit processor scheduling algorithm, Annals of Operations Research. 24 (1990) 233-259.

[9] D. S. Hochbaum, D. B. Shmoys, Using dual approximation algorithms for scheduling problems theoretical and practical results, Journal of the ACM. 34 (1987) 144-162.

[10] S. K. Sahni, Algorithms for scheduling independent tasks, Journal of the ACM. 23 (1976) 116-127.

[11] J. K. Lenstra, D. B. Shmoys and É. Tardos, Approximation algorithms for scheduling unrelated parallel machines, Mathematical programming. 46 (1990) 259-271.

[12] D. G. Kafura, V. Shen, Task scheduling on a multiprocessor system with independent memories, SIAM Journal on Computing. 6 (1977) 167-187.

[13] Y. Azar, J. Naor and R. Rom, The competitiveness of on-line assignments, Journal of Algorithms. 18 (1995) 221-237.

[14] C. A. Glass, H. Kellerer, Parallel machine scheduling with job assignment restrictions, Naval Research Logistics. 54 (2007) 250-257.

[15] J. Ou, J. Y. T. Leung and C. L. Li, Scheduling parallel machines with inclusive processing set restrictions, Naval Research Logistics. 55 (2008) 328-338.

[16] C.-L. Li, X. Wang, Scheduling parallel machines with inclusive processing set restrictions and job release times, European Journal of Operational Research. 200 (2010) 702-710.

[17] C.-L. Li, Q. Li, Scheduling jobs with release dates, equal processing times, and inclusive processing set restrictions, Journal of the Operational Research Society. 66 (2015) 516-523.

[18] C.-L. Li, K. Lee, A note on scheduling jobs with equal processing times and inclusive processing set restrictions, Journal of the Operational Research Society. 67 (2016) 83-86.