Algorithmic Operations Research

# Minimum Cost Flow Problem on Dynamic Multi Generative Networks

Seyed Ahmad Hosseini and Hassan Salehi Fathabadi

Volume 5, Number 1, Spring 2009

URI: https://id.erudit.org/iderudit/aor5_1art05

See table of contents

Publisher(s)

Preeminent Academic Facets Inc.

ISSN

1718-3235 (digital)

Explore this journal

Cite this article

Hosseini, S. A. & Fathabadi, H. S. (2009). Minimum Cost Flow Problem on Dynamic Multi Generative Networks. *Algorithmic Operations Research*, *5*(1), 39–48.

Article abstract

*This paper consists in constructing and modeling Dynamic Multi Generative Network Flows in which the flow commodities is dynamically generated at source nodes and dynamically consumed at sink nodes. It is assumed that the source nodes produce the flow commodities according to k time generative functions and the sink nodes absorb the flow commodities according to k time consumption functions. The minimum cost dynamic flow problem in such networks that extend the classical optimal flow problems on static networks, for a pre-specified time horizon T is defined and mathematically formulated. Moreover, it is showed that the dynamic problem on these networks can be formulated as a linear program whose special structure permits efficient computations of its solution and can be solved by one minimum cost static flow computation on an auxiliary time-commodity expanded network. By using flow decomposition theorem, we elaborate a different model of the problem to reduce its complexity. We consider the problem in the general case when the cost and capacity functions depend on time and commodity.*

*Algorithmic Operations Research Vol.5 (2010) 34–38*

# An improved LS algorithm for the problem of scheduling multi groups of jobs on multi processors at the same speed

Wei Ding and Yi Zhao

Department of Mathematics, Sun Yat-sen University, 510275 Guangzhou, China

**Abstract**

*In the paper we mainly study the $C_{max}$ problem of scheduling $n$ groups of jobs on $n$ special-purpose processors and $m$ general-purpose processors at the same speed provided the ready time of each job is less than $\alpha$ times of its processing time. We first propose an improved $LS$ algorithm. We then show that the bound for the ratio of the approximate solution $T^{LS}$ to the optimal solution $T^*$ is less than $(1+\alpha)(2-\frac{1}{n+m})$. Moreover, we give an example to illustrate that it is tight for any $\alpha \geq 0$.*

*Key words:* Heuristic algorithm, LS algorithm, LPT algorithm, general-purpose processors, special-purpose processors, tight bound.

## 1. Introduction

The problem of scheduling $n$ jobs $\{J_1, J_2, \cdots, J_n\}$ with given processing time on $m$ $(\geq 2)$ identical processors $\{M_1, M_2, \cdots, M_m\}$ with an objective of minimizing the makespan is one of the most well-studied problems in the scheduling literature, where processing job $J_j$ after $J_i$ needs ready time $w(i,j)$. As it has been proved to be $NP-hard$, cf. [10], the study of heuristic algorithms will be important and necessary for this scheduling problem. In fact, hundreds of scheduling theory analysts have cumulatively devoted an impressive number of papers to the worst-case and probabilistic analysis of numerous approximation algorithms for this scheduling problem.

In 1969 Graham [7] showed in his fundamental paper that the bound of this scheduling problem is $2-\frac{1}{m}$ as $w(i,j)=0$ under LS (List Scheduling) algorithm and the tight bound is $\frac{4}{3}-\frac{1}{3m}$ under LPT (Longest Processing Time) algorithm. In 1993 Ovacik and Uzsoy [9] proved the bound is $4-\frac{2}{m}$ as $w(i,j) \leq t_j$, where $t_j$ is the processing time of job $J_j$, under LS algorithm. In 2003 Imreh [8] studied the on-line and off-line problems on two groups of identical processors at different speeds, presented LG (Load Greedy) algorithm, and showed that the bound about minimizing the makespan is $2+\frac{m-1}{k}$ and the bound about minimizing the sum of finish time is $2+\frac{m-2}{k}$, where $m$ and $k$ are the numbers of two groups of identical processors. In 2007, Gairing et al. [6] studied the problem of scheduling one task having $n$ jobs on $m$ processors at different speeds, proposed a combination algorithm about minimal cost stream, and then proved that this algorithm is simple and effective with low complex.

Besides the above well-studied scheduling problem, one may face the problem of scheduling multi groups of jobs on multi processors in real production systems, such as, the problem of processing different types of yarns on spinning machines in spinning mills. Recently, the problem of scheduling multi groups of jobs on multi processors at same or different speeds were studied provided each job has no ready time. In 2004 Ding [1] obtained a tight bound $T^{LPT}/T^* \leq 2$ for the problem of scheduling two groups of jobs on two special-purpose processors at different speeds and $m$ general-purpose processors at same speeds under an improved LPT algorithm provided the speeds of special-purpose processors are faster than those of general-purpose processors. In 2005 Ding [2] gave the bound $T^{LPT}/T^* \leq 4/3$ for the problem of scheduling three groups of jobs on three special-purpose processors and one general-purpose processor at same speeds under an improved LPT algorithm. In the same year Ding [3] got the bound $T^{LPT}/T^* \leq 5/4$ for the problem of scheduling four groups of jobs on four special-purpose processors and one general-purpose processor at same speeds under an improved LPT algorithm. In 2006 Ding [4] studied the

*Email:* Wei Ding [dingwei@mail.sysu.edu.cn], Yi Zhao [stszy@mail.sysu.edu.cn].

problem of scheduling $n$ groups of jobs on one special-purpose processors and $n$ general-purpose processors at same speeds under an improved LPT algorithm, and obtained the bound $T^{LPT}/T^* \leq (n+1)/n$. In 2008 Ding [5] investigated the problem of scheduling $n$ groups of jobs on $n$ special-purpose processors and $m$ general-purpose processors at same speeds under an improved LPT algorithm, and got the bound:

$$\frac{T^{LPT}}{T^*} \leq \begin{cases} \frac{2m+1}{m+1}, & \text{if} \quad m \geq n-1, \\ \frac{m+n}{m+1}, & \text{if} \quad m < n-1. \end{cases}$$

However, if each job has a ready time, then the problem of scheduling multi groups of jobs on multi processors at the same speed has not been studied yet. Note that the LPT algorithm and the improved LPT algorithm are not effective ways to deal with such a problem if each job has a ready time. Meanwhile, the classical LS algorithm is only useful to solve the problem of scheduling one group of jobs on multi processors at same or different speeds. Therefore, our purpose of this study is to propose an improved LS algorithm based on the classical LS algorithm and to use this new algorithm to analyze this scheduling problem provided each job has a ready time.

The remainder of the paper is organized as follows. In Section 2, we propose an improved LS algorithm for this scheduling problem. In Section 3 we obtained the tight bound for this scheduling problem under the improved LS algorithm.

**Notation.** As above and henceforth, we let $L_i$ ($i = 1, \cdots, n$) denote the $i$th group of jobs, and let $M_i$ ($i = 1, \cdots, n$) and $M_{n+j}$ ($j = 1, \cdots, m$) denote the $i$th special-purpose processor and the $j$th general-purpose processor that can process any jobs of any groups, respectively. Let $n_r$ ($r = 1, \cdots, n$) denote the number of jobs in the $r$th group. We then use $J(r,i)$ ($r = 1, \cdots, n$; $i = 1, \cdots, n_r$) to denote the $i$th job of the $r$th group and use $t(r,i)$ ($r = 1, \cdots, n$; $i = 1, \cdots, n_r$) to denote the processing time of $J(r,i)$.

Since the speeds of all processors are the same, for simplicity we assume that the speeds of all processors are equal to 1. If the job $J(h,j)$ ($h = 1, \cdots, n$; $j = 1, \cdots, n_h$) is processed after the job $J(l,i)$ ($l = 1, \cdots, n$; $i = 1, \cdots, n_l$), then we use $w(l,i;h,j)$ to denote the ready time the processor needs.

If the job $J(r,i)$ is assigned to the processor $M_k$ ($k = 1, 2, \cdots, n+m$), then we write $J(r,i) \in M_k$. Let $ML_k$ ($k = 1, 2, \cdots, n+m$) stand for the set of jobs being processed to the processor $M_k$ and let

$$MT_k := \sum_{J(h,j) \in M_k} (w(*,*;h,j) + t(h,j)), \quad k = 1, 2, \cdots, n+m.$$

Then, we write $T^{LS}$ as the actual latest finish time of $n+m$ processors under the improved $LS$ algorithm and $T^*$ as the actual latest finish time of $n+m$ processors under the optimal algorithm, respectively. We finally denote $T^{LPT}$ by the approximate solution under the improved $LPT$ algorithm, $T^{LPT}/T^*$ by the bound of a scheduling problem under the improved LPT algorithm, and $T^{LS}/T^*$ by the bound of a scheduling problem under the improved LS algorithm, respectively.

## 2. An improved LS algorithm

In the section, we will propose an improved LS algorithm for the problem of scheduling multi groups of jobs on multi processors at the same speed provided each job has a ready time.

The algorithm is defined by the fact that whenever a processor becomes idle for assignment, the first job unexecuted is taken from the list and assigned to this processor. If more than one processor is idle, then the algorithm chooses the processor with the smallest index. If the processor is a special-purpose processor for some group, then the first job unexecuted in this group is assigned to the processor. If the processor is a general-purpose processor, then the job with the smallest second index is assigned to the processor. If there are several groups with the same second index, then the job with the smallest first index is assigned. Moreover, there is an arbitrary order for all jobs in any groups at the beginning of being processed.

The steps of the improved $LS$ algorithm are the following:

Step 1. Initialization.
Set $Q_1 = \{1, 2, \cdots, n\}$, $Q_2 = \{n+1, n+2, \cdots, n+m\}$, $i_r = 1$, $1 \leq r \leq n$. Take $ML_r = \emptyset$, $MT_r = 0$, $1 \leq r \leq n+m$.

Step 2. Choose the first idle processor.
If for some $r \in Q_1$, $i_r > n_r$, then set $Q_1 = Q_1 - \{r\}$ (i.e., all jobs in the group $L_r$ have been assigned).
If $Q_1 = \emptyset$, then go to Step 5 (i.e., all jobs in all groups have been assigned).
Set $p = \min\{k' | MT_{k'} = \min_{k \in Q_1 \bigcup Q_2} MT_k\}$ (i.e., seek the first idle processor).

Step 3. Choose the job.

If $p \leq n$, then set $r = p$, $q = i_p$, $i_p = i_p + 1$ (i.e., the special-purpose processor is the first idle processor, then the first job waiting for assignment in the $p$th group is assigned).

If $p > n$ (i.e., the general-purpose processor is the first idle processor), then set $h = \min\{r^{'} | i_{r^{'}} = \min_{r \in Q_1} i_r\}$ (i.e., the job with the smallest second index is assigned), $r = h$, $q = i_h$, $i_h = i_h + 1$.

Step 4. Update the assignment and the latest finish time of the processor $M_p$.

Set $ML_p = ML_p + \{J(r,q)\}$ and $MT_p = MT_p + w(*,*;r,q) + t(r,q)$. Then go to Step 2.

Step 5. Output the assignment $ML_k$, $k = 1, 2, \cdots, n+m$, for every processor and the latest finish time

$$T^{LS} = \max_{1 \leq k \leq n+m} \{MT_k\}.$$

## 3. Analysis of the improved $LS$ algorithm

In the section, we obtain the tight bound for this scheduling problem under the improved LS algorithm in Section 2.

We now present our main theorem.

**Theorem 1.** *Consider the problem of scheduling $n$ groups $\{L_1, L_2, \cdots, L_n\}$ on $n+m$ identical processors $\{M_1, M_2, \cdots, M_{n+m}\}$ at the same speed provided each job has a ready time. Assume that $w(l,i;h,j) \leq \alpha t(h,j)$ for all $l, h, i, j$. Then the bound of this scheduling problem under the improved $LS$ algorithm is*

$$\frac{T^{LS}}{T^*} \leq (1+\alpha)(2 - \frac{1}{n+m}), \qquad \forall \alpha \geq 0.$$

*Proof.* Based on the improved $LS$ algorithm, we may assume that some processor $M_k$ $(1 \leq k \leq n+m)$ is the latest finish processor and some job $J(r,j)$ $(1 \leq r \leq n, 1 \leq j \leq n_r)$ is the latest finish job. Then on the processor $M_k$, we have

$$T^{LS} = MT_k. \tag{1}$$

On other processors, we have

$$\begin{aligned} MT_i &\geq MT_k - (w(*,*;r,j) + t(r,j)) \\ &= T^{LS} - (w(*,*;r,j) + t(r,j)), \\ &\qquad i = 1, 2, \cdots, n+m, \quad i \neq k. \end{aligned} \tag{2}$$

Thus

$$\begin{aligned} \sum_{i=1}^{n+m} MT_i &= MT_k + \sum_{\substack{i=1 \\ i \neq k}}^{n+m} MT_i \\ &\geq (n+m)T^{LS} - (n+m-1)(w(*,*;r,j) \\ &\qquad + t(r,j)). \end{aligned} \tag{3}$$

On the other hand, for the optimal solution $T^*$, we have

$$T^* \geq t(r,j) \tag{4}$$

and

$$T^* \geq \frac{\sum_{l=1}^{n} \sum_{i=1}^{n_l} t(l,i)}{n+m}. \tag{5}$$

By the assumption $w(*,*;r,j) \leq \alpha t(r,j)$ and (4), we get

$$w(*,*;r,j) + t(r,j) \leq (1+\alpha)t(r,j) \leq (1+\alpha)T^*. \tag{6}$$

Then, by (3) and (6), we obtain

$$\sum_{i=1}^{n+m} MT_i \geq (n+m)T^{LS} - (1+\alpha)(n+m-1)T^*. \tag{7}$$

In view of the assumption of the theorem and (5), we deduce

$$\begin{aligned} \sum_{i=1}^{n+m} MT_i &= \sum_{i=1}^{n+m} \sum_{\{t(h,p)\} \in ML_i} (w(*,*;h,p) + t(h,p)) \\ &= \sum_{h=1}^{n} \sum_{p=1}^{n_h} (w(*,*;h,p) + t(h,p)) \\ &\leq (1+\alpha) \sum_{h=1}^{n} \sum_{p=1}^{n_h} t(h,p) \\ &\leq (1+\alpha)(n+m)T^*. \end{aligned} \tag{8}$$

Using (7) and (8), we have

$$\begin{aligned} (1+\alpha)(n+m)T^* &\geq \sum_{i=1}^{n+m} MT_i \\ &\geq (n+m)T^{LS} - (1+\alpha)(n+m-1)T^*. \end{aligned}$$

This implies

$$(1+\alpha)(2n+2m-1)T^* \geq (n+m)T^{LS}.$$

Therefore

$$\frac{T^{LS}}{T^*} \leq \frac{(1+\alpha)(2n+2m-1)}{n+m} = (1+\alpha)(2 - \frac{1}{n+m}).$$

This completes the proof the theorem. □

Next, the following example will show the bound given in Theorem 1 is tight for any $\alpha \geq 0$.

**Example 1.** *Consider the above scheduling problem with* $(n+m)(n+m-1) = kn+r$, $k = p(n+m)+q$, $r < n$, $q < n+m$,

$$L_1 = \{J(1,1), J(1,2), \cdots, J(1,k), J(1,k+1), \cdots, J(1,k+r), J(1,k+r+1)\},$$

$$L_i = \{J(i,1), J(i,2), \cdots, J(i,k)\}, \qquad i = 2, \cdots, n,$$

$t(1,k+r+1) = n+m$, *and other* $t(i,j) = 1$ *(*$i \neq 1$ *and* $j \neq k+r+1$*).*

The schedules for this example under the improved $LS$ algorithm and the optimal algorithm are given in Table 1.

Assume that the job $J(1,k+r+1)$ needs the ready time $\alpha(n+m)$ in the improved $LS$ schedule and that other jobs $J(r,i)$ need the ready time $\alpha$. We adjust the order between the jobs in the optimal schedule (given in Table 2) such that every job needs the ready time $0$. In this example, we have

$$T^{LS} = (1+\alpha)(2n+2m-1), \qquad T^* = n+m,$$

and

$$\frac{T^{LS}}{T^*} = (1+\alpha)(2 - \frac{1}{n+m}).$$

Thus, Example 1 shows that the bound given in Theorem 1 is tight for any $\alpha \geq 0$.

As a consequence of Theorem 1, we have

**Corollary 1.** *If the ready time of every job is* $0$ *in Theorem 1, i.e. all* $w(*,*;*,*) = 0$*, then this scheduling problem under the improved* $LS$ *algorithm has the tight bound*

$$\frac{T^{LS}}{T^*} \leq 2 - \frac{1}{n+m}, \qquad \forall \alpha \geq 0.$$

Received 3-12-2009; revised 24-2-2010; accepted 27-2-2010

**Acknowledgments**

This work was partially supported by NSFC (No. 10971234 and NO. 10671213). The authors thank the referee for valuable comments and suggestions.

## References

[1] W. Ding, Scheduling problem on general purpose machinery and two groups tasks with uniform processors, *Acta Sci. Natur. Univ. Sunyatseni*, **43:2** (2004), 33–36.

[2] W. Ding, A scheduling problem on a general-purpose machine and three groups of tasks with identical processors, *J. Shanghai Univ. Nat. Sci.*, **11:1** (2005), 48–51.

[3] W. Ding, Sequencing of four groups of workpieces on general-purpose machinery, *J. South China Univ. Tech. Nat. Sci.*, **33:10** (2005), 108–111.

[4] W. Ding, A type of scheduling problem on general-purpose machinery and $n$ group tasks, *OR Transactions*, **10:4** (2006), 122–126.

[5] W. Ding, A type of scheduling problem on $m$ general-purpose machinery and $n$ group tasks with uniform processors., *Acta Sci. Natur. Univ. Sunyatseni* , **47:3** (2008), 19–22.

[6] M. Gairing, B. Monien and A. Woclaw, A faster combinatorial approximation algorithm for scheduling unrelated parallel machines, *Theoret. Comput. Sci.*, **380:1-2** (2007), 87–99.

[7] R. L. Graham, Bounds on multiprocessing timing anomalies, *SIAM J. Appl. Math.*, **17:2** (1969), 416–429.

[8] Cs. Imreh, Scheduling problems on two sets of identical machines, *Computing*, **70:4** (2003), 277–294.

[9] I. M. Ovacik and R. Uzsoy, Worst-case error bounds for parallel machine sceduling problems with bounded sequence dependent setup times, *Oper. Res. Lett.*, **14:5** (1993), 251–256.

[10] P. Schuurman and G. J. Woeginger, Polynomial time approximation algorithms for machine scheduling: Ten open problems, *J. Sched.* **2** (1999), 203–213.

Table 1

The improved LS schedule

| Processor | Job1 | $\cdots$ | Job n+m-1 | Job n+m |
|---|---|---|---|---|
| $M_1$ | $J(1,1)$ | $\cdots$ | $J(1,k)$ | $J(1,k+r+1)$ |
| $M_2$ | $J(2,1)$ | $\cdots$ | $J(2,k)$ | |
| $M_3$ | $J(3,1)$ | $\cdots$ | $J(3,k)$ | |
| $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | |
| $M_n$ | $J(n,1)$ | $\cdots$ | $\cdots$ | |
| $M_{n+1}$ | $J(1,2)$ | $\cdots$ | $\cdots$ | |
| $M_{n+2}$ | $J(2,2)$ | $\cdots$ | $\cdots$ | |
| $M_{n+3}$ | $J(3,2)$ | $\cdots$ | $\cdots$ | |
| $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | |
| $\cdots$ | $\cdots$ | $\cdots$ | $J(n,k)$ | |
| $\cdots$ | $\cdots$ | $\cdots$ | $J(1,k+1)$ | |
| $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | |
| $M_{n+m-1}$ | $\cdots$ | $\cdots$ | $J(1,k+r-1)$ | |
| $M_{n+m}$ | $\cdots$ | $\cdots$ | $J(1,k+r)$ | |

Table 2

The optimal schedule

| Processors | Jobs | | | | |
|---|---|---|---|---|---|
| $M_1$ | $J(1,n+m)$ | $J(1,1)$ | $J(1,2)$ | $\cdots$ | $J(1,n+m-1)$ |
| $M_2$ | $J(2,n+m)$ | $J(2,1)$ | $J(2,2)$ | $\cdots$ | $J(2,n+m-1)$ |
| $M_3$ | $J(3,n+m)$ | $J(3,1)$ | $J(3,2)$ | $\cdots$ | $J(3,n+m-1)$ |
| $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $M_n$ | $J(n,n+m)$ | $J(n,1)$ | $J(n,2)$ | $\cdots$ | $J(n,n+m-1)$ |
| $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $M_{n+1}$ $\cdots$ $M_{n+m-1}$ | Note that there are $(n+m)(n+m-1)-n(n+m)=(m+n)(m-1)$ remaining jobs. Then $m-1$ processors process exactly these remaining jobs if each processor $M_j$ is assigned to process $n+m$ jobs. | | | | |
| $M_{n+m}$ | $J(1,k+r+1)$ | | | | |