On the variable common due date, minimal tardy jobs bicriteria two-machine flow shop problem with ordered machines

ALEKSANDAR ILIĆ  
Facebook Inc, Menlo Park, CA, USA  
e-mail: aleksandari@gmail.com  
May 11, 2014

Abstract

We consider the ordinary NP-hard two-machine flow shop problem with the objective of determining simultaneously a minimal common due date and the minimal number of tardy jobs. In [S. S. Panwalkar, C. Koulamas, An \(O(n^2)\) algorithm for the variable common due date, minimal tardy jobs bicriteria two-machine flow shop problem with ordered machines, European Journal of Operational Research 221 (2012), 7–13.], the authors presented quadratic algorithm for the problem when each job has its smaller processing time on the first machine. In this note, we improve the running time of the algorithm to \(O(n \log n)\) using recently introduced modified binary tree data structure. Furthermore, we present \(O(n^2)\) algorithm for the scheduling sequence minimizing makespan for the no-wait variant.

Keywords: scheduling; flow shop; multi-criteria problems; algorithms; binary indexed tree.

1 Introduction

We consider the two-machine flow shop problem with ordered machines in which each job has its smaller processing time on the first machine and with the objective of determining simultaneously a minimal common due date \(d\) and the minimal number of tardy jobs \(n_T\). More precisely, there is a set of \(n\) jobs \(J[i], i = 1, \ldots, n\) all of them are available at time zero; each job \(J[i]\) must be processed non-preemptively and sequentially on two machines \(M_1\) and \(M_2\) with known (integer) processing times \(a[i]\) and \(b[i]\), respectively. Furthermore it holds \(a[i] \leq b[i]\) for all \(i = 1, \ldots, n\). Each machine can process at most one job at a time and the second operation of a job cannot start until the first operation of that job has been completed. Let \(C_1[i]\) and \(C_2[i]\) denote the completion times of job \(J[i]\) on \(M_1\) and \(M_2\), respectively; job \(J[i]\) is tardy if \(C_2[i] > d\), for a given value \(d\). The common objective function is to minimize the maximum completion time \(\max(C_2[i])\) for \(i = 1, \ldots, n\), i.e. the makespan of the job sequence.

Using the three-field notation extended to multi-criteria scheduling problems from [5], the general problem can be denoted as \(F2/d_i = d/d, n_T\) problem and falls into the category
of multi-objective flow shop problems. Therefore, our problem can be denoted as $F_2/a_i \leq b_i, d_i = d/d, n_T$. For the other related problems (like multi-objective flow shop problems, classical flow shop problems with $m$ machines, proportional flow shop problem, ordered flow shop problem, scheduling problems with job rejections) see [4] and references therein.

The state of the art Johnson algorithm [3] yields an optimal arrangement of $n$ jobs on two machines with the minimum completion time $C_{\text{max}}$, by iteratively selecting a job with the shortest processing time and if that is the first machine – schedule the job first, otherwise schedule the job as the last. The Johnson sequence for the $F_2/a_i \leq b_i, k_{\text{jobs}}/C_{\text{max}}$ problem for any $k$ jobs is the shortest processing times (SPT) sequence on $M_1$.

The ordinary NP-hardness of the $F_2/d_i = d/d, n_T$ problem justifies the search for special cases solvable in polynomial time. One such case is when the problem is fully-ordered, that is when the condition $a[i] \leq a[j]$ also implies that $b[i] \leq b[j]$, for each $1 \leq i \leq j \leq n$. This problem was analyzed in [1] in the context of the single-objective $F_2/d_i = d/n_T$ problem in which the common due date is given. T’kindt et al. [6] surveyed the related literature and developed an exact branch and bound algorithm and also a $O(nD^2)$ pseudo-polynomial dynamic programming algorithm for the $F_2/d_i = d/d, n_T$ problem where $D$ is the makespan resulting from applying Johnson’s algorithm to the corresponding maximum completion time problem. The equivalence between $F_2/d_i = d/d, n_T$ and $F_2/k_{\text{jobs}}/C_{\text{max}}$ problems can be easily demonstrated using their single-machine counterparts [4].

The objective of this paper is to show that the problem $F_2/a_i \leq b_i, d_i = d/d, n_T$ is solvable in $O(n \log n)$ time. This problem is equivalent to solving the $F_2/a_i \leq b_i, k_{\text{jobs}}/C_{\text{max}}$ problem for every value $k = 1, \ldots, n$ if only $k$ out of $n$ jobs are retained. This is an optimal algorithm, as it cannot be faster than sorting $n$ numbers. In this note, we improve the proposed quadratic algorithm by Panwalkar and Koulamas from [4] using recently introduced modified binary tree data structure [2]. Furthermore, we also present $O(n^2)$ algorithm for the scheduling specially-structured two-machine flow shop problems with no waiting permitted between the two operations.

2 Optimal algorithm

2.1 Data structure

The binary indexed tree (BIT) is an efficient data structure for maintaining the cumulative frequencies. We will consider the modification of this standard structure to work with minimal/maximal partial summations.

Let $A$ be an array of $n$ elements. The modified binary indexed tree (MBIT) supports the following basic operations with $O(\log n)$ time complexity (for details see [2]):

(i) for given value $x$ and index $i$, add $x$ to the element $A[i], 1 \leq i \leq n$.

(ii) for given interval $[1, i]$, find the sum/min/max of values $A[1], A[2], \ldots, A[i], 1 \leq i \leq n$.

(iii) for given interval $[1, i]$, find the minimum/maximum value among partial sums $A[1], A[1] + A[2], A[1] + A[2] + A[3], \ldots, A[1] + A[2] + \ldots + A[i], 1 \leq i \leq n$.

The operations can be easily extended to return the index where the extremal value is achieved, by storing an additional index data in each node. Furthermore, the comparison can be done in such a way that in case of tie – the maximal index is the leftmost/rightmost one.
2.2 Pivot job and makespan

We assume in the sequel that \( n \) jobs have been renumbered according to this SPT sequence (with ties broken in favor of the shortest \( b[i] \) values).

The makespan is the total length of the schedule jobs \( J \), and this longest path consists of \( n + 1 \) contiguous processing time elements:

\[
C_{\text{max}} = \max_{1 \leq k \leq n} \left( \sum_{i=1}^{k} a[i] + \sum_{j=k}^{n} b[j] \right) = \sum_{i=1}^{n} b[i] + \max_{1 \leq k \leq n} \left( \sum_{i=1}^{k} a[i] - b[i - 1] \right). \tag{1}
\]

We can define new sequence \( c[i] = a[i] - b[i - 1] \) with \( b[0] = 0 \), and also note that the sum of all \( b[i] \) is constant in each iteration. In order to efficiently find the maximal value of the prefix sums of the array \( c \), we can use modified binary indexed tree data structure. Together with the maximal value, we will also store the leftmost index achieving this extremal value in order to determine the pivot job.

If a job \( J[i] \) is removed from the sequence, then the difference between the old makespan and the new makespan will be called the contribution of job \( J[i] \) to the current sequence and will be denoted as \( \delta[i] \). Therefore, we can calculate the contribution of the pivot job in logarithmic time by removing the pivot job, calculating new makespan, and reverting all changes to the data structures.

2.3 Improved algorithm

The proofs of the following results can be found in [4].

**Proposition 1**

(i) For each job \( J[i] \) on the right of the pivot it holds \( \delta[i] = b[i] \).

(ii) For each job \( J[i] \) on the left of the pivot it holds \( \delta[i] = a[i] \), and will not be a candidate for removal as long as the current pivot job and the jobs to the right remain in the sequence.

(iii) Removal of the pivot job will make another job on the right the new pivot job (if exists).

(iv) Removal of any job to the right of the current pivot from the sequence will not change the pivot job and the contributions of any non-pivot jobs.

The pseudo-code of improved PK algorithm from [4] is given in Algorithm 1. The algorithm starts with all jobs sorted as SPT sequence on machine \( M1 \). Then, it identifies the job \( J[i] \) with the maximum contribution \( \delta[i] \) as the candidate job and removes it from the sequence. Once a job is removed, it is not added to the sequence in subsequent iterations from 1 to \( n \). It should be pointed out that the PK Algorithm emulates the action of the optimal algorithm for the corresponding single-machine problem.

In order to speed up the algorithm, we are going to maintain two MBITs for storing the maximal suffix values of \( b \) and the maximal prefix partial sums of \( c \). Note that there is no need for storing the maximums of the array \( a \), as the array \( a \) is sorted and \( a[i] \leq b[i] \) holds for all \( 1 \leq i \leq n \). We will also store the sum of all \( b[i] \) in the current sequence and use it in the equation (1).
We first construct the data structures in $O(n \log n)$ time and update them as we remove the jobs from the sequence. The leafs of these tree structures will contain the arrays $b$ and $c$. When the job $J[i]$ is removed, we simply set $b[i] = 0$ in $maxB$ modified binary indexed tree - and all queries will return correct indices as $b[i] > 0$ holds for all existing jobs.

Removal of the job $J[i]$ will involve updating the numbers $c[i]$ and is slightly more complicated. We need to maintain two arrays $left$ and $right$ which will contain the indices of the first remaining jobs to the left and right, respectively. At the beginning, it holds $left[i] = i - 1$ and $right[i] = i + 1$ for $1 \leq i \leq n$. When the job $J[i]$ is removed, we need to update the values $c[left[i]]$ and $c[right[i]]$ and make $c[i] = 0$. Furthermore, $right[left[i]] = right[i]$ and $left[right[i]] = left[i]$. In the modified binary tree structure, we will always store the index of the leftmost value which will be ensure to always find the existing jobs.

Algorithm 1: Calculating the optimal job scheduling.

Input: The job sequence $J$ with execution times $a[i]$ and $b[i]$.

Output: The order of jobs.

1. Sort the jobs by $a[i]$ and in case of tie by $b[i]$;
2. Create MBIT $maxB$ for the maximal suffixes of $b$;
3. Create MBIT $maxC$ for the maximal prefix sums of $c[i] = a[i] - b[i - 1]$;
4. Create the arrays $left$ and $right$;
5. for $k = 1$ to $n$ do
6.   Calculate the makespan $C_{max}$ and the pivot job $p$ of the current sequence of jobs;
7.   Using $maxC$, find the contribution of the pivot job $p$, by removing and restoring the pivot job from the current sequence;
8.   Using $maxB$, find the job $i$ with the maximal contribution: max of $\delta[p], b[p + 1], \ldots, b[n]$;
9.   Remove the job $i$ and update data structures $maxB$ and $maxC$, and arrays $left$ and $right$;
10. end

Therefore, the preprocessing is taking $O(n \log n)$ time, and each operation in the for loop is $O(\log n)$ time. We conclude this section with the following proposition.

Proposition 2 The described algorithm (always removing the job with the highest contribution) is optimal for the $F2/a_i \leq b_i, k\text{jobs}/C_{max}$ and $F2/a_i \leq b_i, d_i = d/d, n_T$ problems with the time and space complexity $O(n \log n)$.

The algorithm enumerates the $n + 1$ Pareto optima for each one of these two problems in $O(n \log n)$ time. For the future work, it would be interesting to extend the current approach to other specially-structured flow shop problems with two or more machines and improve the existing flow shop scheduling algorithms using more efficient data structures.

3 No wait flow shop problem

In this section we will consider specially-structured two-machine flow shop problems with no waiting permitted between the two operations with the objective to minimize the makespan $C_{max}$, with the same conditions $a[i] \leq b[i]$. 

4
Consider a graph $G$, where the vertices correspond to the jobs and a directed edge $(i, j)$ exists if and only if $a[i] \leq b[j]$. Note that in $G$ there can exists both edges $(i, j)$ and $(j, i)$ and the graph also can contain cycles. We can perform a breadth first search from any vertex in $G$ in $O(n^2)$ time, and find the shortest paths from any vertex in the graph.

Let us define a chain of jobs as maximal subsequence of jobs $J[i], J[i+1], \ldots, J[j]$, for which it holds $a[k+1] \leq b[k]$ for $k = i, i+1, \ldots, j-1$. Each job sequence $J$ can be splitted into multiple chains, and each chain has a makespan of $a[i] + \sum_{k=i}^{j} b[k]$.

Reorder the jobs by the longest processing times, i.e. assume that $b[1] \geq b[2] \geq \ldots \geq b[n]$. Then the whole sequence is one large chain as $a[i+1] \leq b[i+1] \leq b[i]$ for $i = 1, 2, \ldots, n-1$, and the makespan of this sequence is equal to

$$a[1] + \sum_{i=1}^{n} b[i].$$

One can put some of the jobs in front of the first job $J[1]$ in order to reduce the contribution $a[1]$. Let $s$ be the job with the smallest value $a[s]$ among all jobs reachable from the vertex $J[1]$ in the graph $G$ (following the reverse directions of the edges in $G$). Then,

$$C_{\text{max}} \geq a[s] + \sum_{i=1}^{n} b[i],$$

as we can put the jobs from the path from $J[1]$ to $J[s]$ at the beginning and remove them from the second part.

Now consider the optimal no-wait job sequence minimizing the makespan and let the job $J[1]$ be on the position $k$. If there are multiple chains, one can easily remove the chains that do not contain $J[1]$ and merge them after the position $k$ by sorting all jobs. That way the overall makespan is reduced. Therefore, there can be only one chain and the actual minimum is achieved by having the job $a[s]$ at the first position.

**Proposition 3** The minimal makespan for the no-wait flow shop two machine problem is equal to $C_{\text{max}} = a[s] + \sum_{i=1}^{n} b[i]$, where $J[s]$ is the job with the smallest $a[s]$ reachable from the job $J[1]$ with the maximal $b[i]$ value in the underlying graph $G$. The time complexity of the scheduling algorithm is $O(n^2)$.

We leave for the future research to see if this approach can be extended when the objectives are to simultaneously determine a minimal common due date and the minimal number of tardy jobs.

**References**

[1] F. Della Croce, J. N. D. Gupta, R. Tadei, *Minimizing tardy jobs in a flow shop with common due date*, European Journal of Operational Research *120* (2000) 375–381.

[2] A. Ilić, *Efficient algorithm for the vertex connectivity of trapezoid graphs*, Information Processing Letters *113* (2013) 398–404.

[3] S. M. Johnson, *Optimal two and three stage production schedules with setup times included*, Naval Research Logistics Quarterly *1* (1954) 61–68.
[4] S. S. Panwalkar, C. Koulamas, An $O(n^2)$ algorithm for the variable common due date, minimal tardy jobs bicriteria two-machine flow shop problem with ordered machines, European Journal of Operational Research 221 (2012) 7–13.

[5] V. T’kindt, J. C. Billaut, Multi-criteria scheduling: theory, models and algorithms, Springer-Verlag (2002), Heidelberg, Germany.

[6] V. T’kindt, F. Della Croce, J. L. Bouquard, Enumeration of Pareto optima for a flow shop scheduling problem with two criteria, Informs Journal on Computing 19 (2007) 64–72.