An Efficient Implementation of a Two-stage Production Scheduling Algorithm

WING S. CHOW
Department of Administrative Information Management, Hong Kong Baptist College, Hong Kong

In this paper, a most efficient implementation of Johnson's two-stage algorithm is presented. An illustration with numerical example and computational results is also included.

Key words: production, scheduling, sequencing

One of the most difficult problems arising in implementing scheduling algorithms in the real world is their high computational complexity. To solve this problem, research to investigate the efficient implementation of the literature should be conducted. The objective of this paper is to serve this purpose, and the problem to be considered is Johnson's two-stage production scheduling algorithm.

The objective of Johnson's two-stage algorithm is to determine a schedule of $n$ jobs, $S(n) = (J_1, J_2, \ldots, J_n)$, process through two machines, each job in the order of $M_1$, $M_2$, so that the maximum flow time is minimized. In the literature of scheduling, this problem is referred to as the $n/2/F/F_{\text{max}}$ problem. In order to discuss Johnson's algorithm in detail, let $a_i$ and $b_i$ be the processing time of the $i$th job ($J_i$) on the first and second machine, respectively. Then the algorithm presented by Johnson$^1$ can be summarized as follows:

Step 1
Obtain two lists from $S(n) = (J_1, J_2, \ldots, J_n)$:
List 1: Sort $n$ job numbers and corresponding machine times (i.e. $a_i$ and $b_i$) according to the ascending value of $a_i$.
List 2: Sort $n$ job numbers and corresponding machine times (i.e. $a_i$ and $b_i$) according to the ascending value of $b_i$.

Step 2
Construct the optimal schedule by comparing the first $a_i$ value of list 1 with the first $b_i$ value of list 2. If the $a_i$ value of list 1 is smaller than or equal to $b_i$, the value of list 2, then place the corresponding job number of $a_i$ in list 1 in the first position; otherwise place the corresponding job number of $b_i$ in list 2 in the last position.

Step 3
Repeat step 2 until all jobs are assigned.

Since the computational complexity of the optimal sorting algorithm is $O(n \log n)$ (see Horowitz and Sahni$^2$), it is easy to see that the computational complexity of the above algorithm is $O_J = O(2n \log n) + O(n)$, i.e. two sorting operations in step 1 and an assignment task in step 2. Kusiak$^3$ has subsequently simplified this algorithm by replacing it with one that included three operations: one for sorting the data set and the other two for clustering and assigning $n$ jobs into a proper sequence. Its computational complexity is $O_K = O(n \log n) + O(2n)$.

Before a more efficient implementation of this algorithm can be presented, the following rules and lemmas are established. Decompose $n$ jobs, $S(n) = (J_1, J_2, \ldots, J_n)$, into the following two lists:
List 1: $S(\bar{x})$, $\forall J_i$ such that $a_i \leq b_i$, $i = 1, 2, \ldots, n$;
List 2: $S(\bar{y})$, $\forall J_i$ such that $a_i > b_i$, $i = 1, 2, \ldots, n$.

1049
**Lemma 1**

Scheduling jobs of $S(\bar{x})$ according to the ascending value of $a_i$ provides a minimum makespan schedule, $S^*(\bar{x})$.

**Lemma 2**

Scheduling jobs of $S(\bar{y})$ according to the descending value of $b_j$ provides a minimum makespan schedule, $S^*(\bar{y})$.

**Lemma 3**

The minimum makespan schedule for $S(n)$ is as follows:

$$S^*(n) = \{S^*(\bar{x}), S^*(\bar{y})\}.$$ 

Proofs of these lemmas are similar to those shown by Johnson.\(^1\) Based on the above lemmas, the following algorithm is presented.

**ALGORITHM**

**Step 1**

Decompose $n$ jobs, $S(n) = (J_1, J_2, \ldots, J_n)$, into one of the following lists:

- List 1: $S(\bar{x})$, $\forall J_i$ such that $a_i \leq b_i$, $i = 1, 2, \ldots, n$;
- List 2: $S(\bar{y})$, $\forall J_i$ such that $a_i > b_i$, $i = 1, 2, \ldots, n$.

**Step 2**

(i) Sort job numbers in $S(\bar{x})$ according to the ascending value of $a_i$, and denote the results as $S^*(\bar{x})$.

(ii) Sort job numbers in $S(\bar{y})$ according to the descending value of $b_j$, and denote the result as $S^*(\bar{y})$.

**Step 3**

The optimal schedule of $S(n)$ is

$$S^*(n) = \{S^*(\bar{x}), S^*(\bar{y})\}.$$ 

To determine the computational complexity of the above algorithm, let the number of jobs in $S(\bar{y})$ be $k$; then its complexity can be computed as follows:

<table>
<thead>
<tr>
<th>Step number</th>
<th>Computational complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>2</td>
<td>$O([n - k] \log([n - k])) + O(k \log k)$</td>
</tr>
<tr>
<td>3</td>
<td>$O_k = O([n - k] \log([n - k])) + O(k \log k) + O(n)$</td>
</tr>
</tbody>
</table>

Comparing the above computational complexity with those of $O_J$ and $O_K$, one can see that $O_k < O_k < O_J$. For example, let $k = n/2$; then $O_k = O(n/2 \log n/2) + O(n/2 \log n/2) + O(n) < O_K < O_J$.

**A NUMERICAL ILLUSTRATION**

The above algorithm is illustrated in example 1, where data is obtained from the example provided by Johnson.\(^1\)
Example 1

Solve the $S/2/F/F_{\text{max}}$ scheduling problem for the following set of data:

<table>
<thead>
<tr>
<th>Job number</th>
<th>$a_i$</th>
<th>$b_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$J_2$</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>$J_3$</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>$J_4$</td>
<td>6</td>
<td>30</td>
</tr>
<tr>
<td>$J_5$</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Step 1

The above list of $S(n)$ is decomposed into:
List 1: $S(\bar{x}) = (J_1, J_4, J_5)$;
List 2: $S(\bar{y}) = (J_2, J_3)$.

Step 2

Sort the $a_i$ processing time of list 1 and the $b_i$ processing time of list 2 in ascending and descending order, respectively, providing the following results:

$S^*(\bar{x}) = \{J_5, J_1, J_4\}$
$S^*(\bar{y}) = \{J_3, J_2\}$.

Step 3

The optimal schedule is

$S^*(n) = \{S^*(\bar{x}), S^*(\bar{y})\}$

$\quad = \{J_5, J_1, J_4, J_3, J_2\}$.

The makespan of the above schedule is 47 time units.

COMPUTATIONAL EXPERIENCE

To test the performance of the proposed and the original algorithm, VAX FORTRAN codes were developed, and a number of problems of different sizes have been solved on a VAX 785 computer. The data for these problems was generated based on the uniform random number generator. The initial seed value is assigned as 8749, and values of $a_i$ and $b_i$ are in the range of [0, 1000) time units. The computational results are summarized in Table 1.

<table>
<thead>
<tr>
<th>Number of jobs, $n$</th>
<th>CPU (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Proposed</td>
</tr>
<tr>
<td>100</td>
<td>0.02</td>
</tr>
<tr>
<td>300</td>
<td>0.17</td>
</tr>
<tr>
<td>500</td>
<td>0.48</td>
</tr>
<tr>
<td>700</td>
<td>0.98</td>
</tr>
<tr>
<td>1000</td>
<td>1.96</td>
</tr>
<tr>
<td>1500</td>
<td>8.52</td>
</tr>
<tr>
<td>2000</td>
<td>17.27</td>
</tr>
<tr>
<td>2500</td>
<td>27.39</td>
</tr>
<tr>
<td>3000</td>
<td>39.33</td>
</tr>
<tr>
<td>3500</td>
<td>54.14</td>
</tr>
</tbody>
</table>

The computational experience showed that the proposed algorithm is very efficient. To show the efficiency of this algorithm, consider $n = 3500$. It takes only 54 sec to obtain the optimal schedule. To solve the same size of problem using the original algorithm, more than 163 sec are required.
CONCLUSION

This paper proposed an efficient implementation of Johnson’s two-stage algorithm. The effectiveness of the proposed algorithm was theoretically proven and empirically evaluated. It was shown that the proposed algorithm is particularly advantageous for problems with a large number of jobs to be scheduled. Furthermore, it should also be clear that the proposed algorithm can be applied to solve $n/3/F/F_{\text{max}}$, which was discussed by Johnson.\footnote{3}

REFERENCES