Uniform Parallel Machine Scheduling with Dedicated Machines, Job Splitting and Setup Resources*

Hyun-Jung Kim1 and Jun-Ho Lee2

Abstract—This paper examines a uniform parallel machine scheduling problem in which jobs can be split arbitrary into multiple sections and such job sections can be processed on a set of dedicated machines simultaneously. Once a job type is changed, a setup performed by an operator is required. The setup time is sequence-independent, and the number of setup operators is limited. Machines conduct the same operation but have different speeds. The objective is to minimize the maximum completion time. This problem is motivated from real-life applications that manufacture automotive pistons in Korea. We propose efficient heuristic algorithms for this problem and show experimentally that the performance of the algorithms is good enough to be used in practice.

I. INTRODUCTION

In uniform parallel machine scheduling, \( n \) jobs are processed on \( m \) machines in parallel with different speeds. The speed of machine \( i \) where \( i \in M, M = \{1, 2, \ldots, m\} \) is denoted by \( v_i \), and job \( j \) where \( j \in N, N = \{1, 2, \ldots, n\} \) has the processing time of \( \frac{p_{ij}}{v_i} \) when job \( j \) receives all of its processing from machine \( i \). When machines have the same speed, i.e., \( v_i = 1 \) for all \( i \), the environment is the same as identical parallel machine scheduling. Job \( j \) can only be processed on a set of designated machines, \( M_j \), and can be split into multiple sections that can be processed on several machines simultaneously. Machine \( i \) can process only jobs in \( J_i \) where \( J_i = \{j \mid i \in M_j\} \). When a job type is changed on a machine, a setup is performed by one of \( r < m \) operators that are not sufficient to set up all of \( m \) machines at the same time. The setup time for job \( j, s_j \), is sequence-independent. The objective is to minimize the maximum completion time, \( C_{max} \). In this paper, we examine this uniform parallel machine scheduling problem with dedicated machines, job splitting, and limited setup resources, that can be denoted as \( Qm|s_j, M_j, r, split|C_{max} \) [1]. This problem is motivated from real systems that manufacture automotive pistons, bolts and nuts for automotive engines, fan or equipment filter units, textiles, printed circuit boards, and network computing [2].

II. LITERATURE REVIEW

For uniform parallel machine scheduling, Lee et al. [3] developed two heuristic algorithms to derive an optimal assignment of operators to machines with learning effects in order to minimize the makespan. Elvikis et al. [4] considered a uniform parallel machine scheduling problem with two jobs that consist of multiple operations, and derived Pareto optima with makespan and cost functions. For a job splitting property, Yalaoui and Chu [2] proposed an efficient heuristic algorithm for parallel machine scheduling with sequence-dependent setup times and evaluated its performance with proposed lower bounds. Wang et al. [5] examined a parallel machine scheduling problem with job splitting and learning with the total completion time measure, and used a branch and bound algorithm for small-sized problems and heuristics for large-sized ones. Yeh et al. [6] proposed several metaheuristic algorithms for uniform parallel machine scheduling given that some resource consumption cannot exceed a certain level. These studies assume that resources are required to process jobs on machines. Even though there have been numerous papers on parallel machine scheduling, no study has been performed for our problem; uniform parallel machine scheduling with dedicated machines, job splitting, and setup resource constraints.

III. PROBLEM DESCRIPTION & ANALYSIS

The problem we consider can be easily proven to be NP-hard because a parallel machine scheduling problem with two machines, which is a special case of our problem with \( v_i = 1 \) for all \( i \in M, r = m, s_j = 0 \) for all \( j \in N \), and \( M_j = M \) for all \( j \in M \), is proven to be NP-hard [1]. Hence, we derive efficient heuristic algorithms.

We assume that jobs processed on machines for the first time do not require setup operations. This is because once all jobs are completed for a given period, mostly one or several weeks, preventive maintenance for machines is performed, and then they are set up for the next desired states [2]. Setup times are not affected by speeds of machines since they are performed by setup operators. The lengths of jobs processed on each machine and their sequence should be determined by considering setup resources and dedicated machines with different speeds in order to minimize the makespan.

We first define \( N_1 \) as a job set that contains \( m \) jobs in which \( \sum_{j \in N_1} s_j \) is maximized and those \( m \) jobs in \( N_1 \) can be assigned to \( m \) machines one by one while satisfying dedicated machine constraints, \( M_j \). Suppose that there are two machines and three jobs, and \( s_1, s_2, \) and \( s_3 \) are 2, 1, and 3, respectively. If \( M_1 = M_3 = \{1\} \) and \( M_2 = \{1, 2\}, N_1 \)
contains jobs 2 and 3 instead of jobs 1 and 3 even though $s_2 + s_3 < s_1 + s_3$ because jobs 1 and 3 must be processed on machine 1. A job set $N_2$ includes jobs that are in $N \backslash N_1$. $N_1$ can be obtained by using the Hungarian method [7]. We let $S_l$ indicate a set of jobs that have the same set of dedicated machines, i.e., $M_j = M_k$ if jobs $j$ and $k$ are in $S_l$. In the above example, jobs 1 and 3 are in the same set. We then have the following four lower bounds of our problem.

**Lemma 1:** A job-based lower bound $LB_1$ is

$$
\text{max}_{j \in S_l} \sum_{i \in M_j} v_{i}.
$$

**Lemma 2:** A machine-based lower bound $LB_2$ is

$$
\sum_{j \in N} p_j + \sum_{j \in N_2} s_j.
$$

**Lemma 3:** A resource-based lower bound $LB_3$ is

$$
\sum_{j \in S_l' \backslash S_l} v_{j}.
$$

**Lemma 4:** A job set-based lower bound $LB_4$ is

$$
\max_{S_l} \left\{ \sum_{j \in S_l} p_j + \sum_{j \in S_l' \backslash S_l} s_j \right\} \text{where } M_j \text{ is a set of machines that can process jobs in } S_l, \text{ and } S_l' \text{ is a set of jobs in } S_l \text{ except for } |M_{S_l}| \text{ jobs with the largest setup times among jobs in } S_l.
$$

**Theorem 1:** A lower bound on the makespan of the problem, $LB$, is $\max\{LB_1, LB_2, LB_3, LB_4\}$.

**IV. HEURISTIC ALGORITHMS**

Since no setup is required for the first jobs on machines, assigning jobs with large setup times as the first ones can lead to the makespan reduction. The optimal assignment of $m$ jobs to $m$ machines can be found with the Hungarian method. In the method, $n - m$ dummy machines are made, and the setup times of $n$ jobs on those machines are set to 0. The setup time of job $j$ on machine $i$, $i \notin M_j$ is also set to 0. Then the Hungarian method is applied with $n$ jobs and $n$ machines. After assigning $m$ jobs, each time a machine becomes available, a job is chosen according to some of well-known priority rules. The first one is the least flexible job (LFI) rule. When machine $i$ finishes processing a job, the job with the smallest $|M_j|$ among jobs in $J_i$ is chosen and assigned. The LFI rule performs well for dedicated parallel machine scheduling. The second one is the longest processing time (LPT) rule that selects the job with the largest processing time among jobs in $J_i$. The rule is good for the makespan measure because it balances the loads of parallel machines. After assigning all of $n$ jobs, the loads of machines are balanced by splitting the last job on each machine into two sections and assigning one section to another machine by considering different speeds and dedication of machines.

We now propose a heuristic algorithm that combines the Hungarian method for assigning first $m$ jobs, one of priority rules (LFI or LPT) for assigning $n - m$ jobs and the load balancing step on machines. We test the above two priority rules and show their performance in Section V. Let $L_i$ be the completion time of the last job on machine $i$.

**Algorithm I**

- Step 1: Using the Hungarian method, select $m$ jobs and assign them to each machine so that the sum of setup times of such $m$ jobs is maximized, and define a set $N_1$ that contains these $m$ jobs.
- Step 2: Assign $n - m$ jobs in $N_2$ where $N_2 = N \backslash N_1$ according to the LFI or LPT rule and update $L_i$ \( \forall i \).
- Step 3: Select machine $l$ where $l = \arg \max_{i \in M} L_i$. If $\text{maxReducibleTime}(l) > 0$, assign a section of the last job on machine $l$ to machine $\text{bestAssignableMachine}(l)$ so that $L_i = L_i \ast$, where $i = \text{bestAssignableMachine}(l)$, and repeat Step 3. Otherwise, go to Step 4.
- Step 4: Consider $M_k$ where $l = \arg \max_{i \in M} L_i$ and $k_l$ is the last job on machine $l$. For each machine $i \in M_k$, update $L_i$ to $L_i - \text{maxReducibleTime}(i)$ temporarily, and store its original value. If $\text{maxReducibleTime}(l) = 0$, stop. Otherwise, let $j = \text{bestAssignableMachine}(l)$ and restore $L_i$ to its original value. Assign a section of the last job on machine $j$ to machine $k$ where $k = \text{bestAssignableMachine}(j)$ so that $L_j = L_k$, and go to Step 3.

**Function: maxReducibleTime($l$)**

For a given machine $l$ and its last job $k_l$, return $\Delta$ where

$$\Delta = \max_{i \in M_{k_l}} \left\{ \frac{L_i - L_{k_l} - v_{i,k_l}}{1 + \frac{\alpha}{p_i}} \left\{ \begin{array}{ll} 0 & \text{if } v_{i,k_l} \leq 0 \end{array} \right. \right\} \right.$$.

**Function: bestAssignableMachine($l$)**

For a given machine $l$ and its last job $k_l$, return machine $i$ where $i = \arg \max_{i \in M_{k_l}} \left\{ \frac{L_i - L_{k_l} - v_{i,k_l}}{1 + \frac{\alpha}{p_i}} \right\}$.

With Algorithm I, we can obtain a feasible and effective solution. The solution is updated further by splitting jobs with longer processing times at a time in Algorithm II.

**Algorithm II**

- Step 1: Let both $N$ and $N^*$ be the initial job list and $C_{max} = \infty$.
- Step 2: Apply Algorithm I with jobs in $N$ and update $C_{max}$ to the resulting makespan. If $C_{max}$ is equal to the lower bound, stop. Otherwise, go to Step 3.
- Step 3: If $N^* = \emptyset$ or all of the jobs in $N^*$ have the processing time less than $\epsilon$ ($\epsilon$ is a very small positive real number to avoid an infinite loop), stop. Otherwise, select job $l$ in $N^*$ according to the LPT rule. Split job $l$ into two sections, $l_1$ and $l_2$, with the same processing time. Apply Algorithm I with jobs in $(N \backslash l) \cup \{l_1, l_2\}$. Let the makespan obtained with the updated $N$ be $C'_{max}$.
- Step 4: If $C'_{max} < C_{max}$, set $C_{max} = C'_{max}$ and update both $N$ and $N^*$ by eliminating job $l$ and adding jobs $l_1$ and $l_2$. Otherwise, update $N^*$ by eliminating $l$. Go to Step 3.

**V. EXPERIMENTS**

The proposed algorithms are tested with various scenarios. The machine speed is determined randomly between 0.8 and 1.2, and the setup times for jobs are generated with $\alpha p_j$ where $\alpha$ is selected randomly within $[0.01, 0.1], [0.1, 0.2]$, and $[0.1, 0.5]$. Processing times are generated between 10 and 100. There are three levels of machine dedications, high, medium (i.e., mid), and low, and in each level, jobs can be
processed on a machine with the probability of 50 %, 50 %–90 %, and 90 %, respectively. Fig. 1 shows experimental results with $m$ of 5 and $n$ of 40 for Algorithm I with LFJ (A1 (LFJ)), Algorithm I with LPT (A1 (LPT)), Algorithm II with LFJ (A2 (LFJ)), and Algorithm II with LPT (A2 (LPT)). For each problem with a certain range of setup times, a dedication level, and the number of resources, 100 instances are generated, and the average gaps are shown in Fig. 1; there are 45 types of problems from 5 resources, 3 setup time ranges, and 3 dedication levels. Hence, 4500 instances are tested for each algorithm. The gaps from lower bounds, computed as (makespan from the proposed algorithm - lower bound)/lower bound, are represented in Fig. 1. The four graphs in Fig. 1 represent the average gaps of the proposed four algorithms depending on $r$, $s_j$, dedication levels, and $n$, respectively. In Fig. 1(a), A1 (LPT) performs better than A1 (LFJ) only when $r$ is 1 whereas A2 (LPT) works better than A2 (LFJ) in most instances except for $r$ of 5. The gaps of A2 (LFJ) and A2 (LPT) are improved as much as 21 % and 74 % from A1 (LFJ) and A1 (LPT), respectively. In Fig. 1(b), the average gaps tend to increase as the setup time variations become large. When the variations are small, $\alpha \in [0.01, 0.1]$ and $\alpha \in [0.1, 0.2]$, A1 (LFJ) performs better than A1 (LPT) but A1 (LFJ) is outperformed when $\alpha \in [0.1, 0.5]$. In Fig. 1(c), we can see that, in the high dedication level, the performance of A1 (LFJ) and A2 (LFJ) is good but A1 (LPT) does not work well. When the dedication level is not high, A2 (LPT) provides the best performance. Fig. 1(d) shows the results with $n$ of 40, 60, and 80 for each algorithm. As $n$ increases, the average gaps tend to be decreased. The overall average gaps of the four algorithms with $m$ of 5 are 2.96, 2.94, 2.51, and 2.08 %, respectively. For $m$ of 10, the average gaps of the four algorithms are 5.63, 3.77, 4.97, and 3.00 %, respectively. In addition, all instances are solved within one second.

VI. CONCLUSION

We have analyzed the uniform parallel machine scheduling problems with dedicated machines, job splitting, and setup resources. We have derived four lower bounds of the problem and proposed heuristic algorithms. We then showed the experimental results with different setup time ranges, resources, and dedication levels. The average gap was less than 3 % with A2 (LPT). More scenarios should be tested with the proposed algorithms. In addition, sequence-dependent setup times need to be further analyzed in our problem.

REFERENCES

[1] M. L. Pinedo, Scheduling Theory, Algorithms, and Systems. New York, NY, USA: Springer, 2012.
[2] F. Yalaoui and C. Chu, “An efficient heuristic approach for parallel machine scheduling with job splitting and sequence-dependent setup times,” IIE Transactions, vol. 35, no. 2, pp. 183–190, 2003.
[3] W.-C. Lee, M.-C. Chuang, and W.-C. Yeh, “Uniform parallel-machine scheduling to minimize makespan with position-based learning curves,” Computers & Industrial Engineering, vol. 63, no. 4, pp. 813–818, 2012.
[4] D. Elvikis, H. W. Hamacher, and V. T’kindt, “Scheduling two agents on uniform parallel machines with makespan and cost functions,” Journal of Scheduling, vol. 14, no. 5, pp. 471–481, 2011.
[5] C. Wang, C. Liu, Z.-H. Zhang, and L. Zheng, “Minimizing the total completion time for parallel machine scheduling with job splitting and learning,” Computers & Industrial Engineering, vol. 97, pp. 170–182, 2012.
[6] W.-C. Yeh, M.-C. Chuang, and W.-C. Lee, “Uniform parallel machine scheduling with resource consumption constraint,” Applied Mathematical Modelling, vol. 39, no. 8, pp. 2131–2138, 2015.
[7] H. W. Kuhn, “The Hungarian method for the assignment problem,” Naval Research Logistics, vol. 2, no. 1-2, pp. 83–97, 1955.