Advances in Intelligent Systems Research, volume 159
International Conference on Mathematics, Modelling, Simulation and Algorithms (MMSA 2018)

# Minimizing the Maximum Flow Time for Flexible Job Shop Problem with Parallel Machines Considering Release Time

Xi Xiang[1], Changchun Liu[2] and Lixin Miao[1,*]
[1]Division of Logistics and Transportation, Graduate School at Shenzhen, Tsinghua University, Shenzhen 518055, China
[2]Department of Industrial Engineering, Tsinghua University, Beijing 100084, China
[*]Corresponding author

***Abstract*—This paper studies a flexible job shop problem with parallel machines considering release time. The objective aims to minimize the maximum flow time. A property which can reduce the dimension of solution space is proposed. A branch-and-bound algorithm is proposed to solve the problem. Through numerical experiments, the proposed algorithm is proved to be both effective and efficient in solving this flexible job shop problem.**

***Keywords—flexible job shop problem; parallel machines; flow time***

## I. INTRODUCTION

A classic job shop problem (JSP) has a set of jobs processed by a set of machines. Each job is processed on machines in a given sequence with a given processing time, and each machine can process only one type of operation. However, along with the rapid industrialization, a machine may have the flexible capability to be set up to process more than one type of operations in modern manufacturing plant. Therefore, a modified version called a flexible JSP (FJSP) is generated. Brucker and Schlie (1990) concluded that there are two types of FJSP. One is that jobs have alternative operation sequences and alternative identical or non-identical machines for each operation. This kind of problem aims to determine the operation sequences for jobs and the job processing orders on machines. The other one is that jobs can have only fixed operation sequences but alternative identical or non-identical machines for each operation. This kind of problem is to arrange jobs to machines according to their fixed operation sequences. This paper studies the second kind of problem which arrange jobs to alternative identical machines according to their fixed operation sequences.

In this problem, jobs visit a certain machine or a set of machines. In addition, each operation is processed by one machine and the processing time is the same for any of the identical parallel machines. In this paper, a setup depends only on the job to be processed and the machine processing it, hence the setup time can be seen as included in the processing time. The transportation time is ignored in this study.

In previous articles, the objective always aims to minimize the maximum makespan( $C_{\max}$ ). However, such an objective will generate an extreme case that a job with an early release time has a late completion time, which means such a job has a long flow time and this case seems unreasonable. Therefore, we aim to minimize the maximum flow time ( $F_{\max}$ ) in this paper. To solve the problem, a branch-and-bound algorithm is proposed.

## II. LITERATURE REVIEW

There are numerous papers studied on FJSP which was first proposed by Brucker and Schlie (1990). Brucker and Schlie (1990) studied the assignment and scheduling problems in FJSP with two jobs and proposed a polynomial algorithm to solve it. For the first type FJSP, Baykasoglu (2002) proposed a linguistic based meta-heuristic modeling and solution approach and Choi and Choi (2002) formulated a mixed-integer model and proposed a local search algorithm for alternative operations and sequence-dependent setups in various manufacturing environments. In order to balance machine loads and minimize the makespan and mean flow time, Kim et al. (2003) proposed a new approach to solve process planning and scheduling problem simultaneously. Later, Chen et al. (2008) developed a heuristic algorithm and employed it to solve real industrial FJSP. Balin (2011) proposed new operations in genetic algorithm (GA) to adapt non-identical parallel machine scheduling problem in order to minimum the makespan. Liu et al. (2011) presented an adaptive annealing genetic algorithm to deal with the job-shop planning and scheduling problem for the single-piece, small-batch, custom production mode. Moradi et al. (2011) proposed four algorithms to solve FJSP with preventive maintenance activities under the multi-objective optimization approaches, and the performance of the approaches is investigated using a benchmark with a large number of test instances.

For the second type FJSP, Norman and Bean (1999)} developed a GA-based approach to minimize total tardiness considering some practical constraints. Mastrolilli and Gambardella (2000) proposed a neighborhood function using tabu search method incorporating two neighborhood functions to minimize the makespan. To solve resource-constrained FJSP problem with multiple identical machines, Chan et al. (2006) proposed a GA-based approach. Chan et al. (2011) presented a GA-based job shop scheduler for a flexible multi-product,

parallel machine sheet metal job shop with an objective to minimum makespan.

## III. NOTATIONS

A typical FJSP involves a set of jobs, *I={1,…,n}*, and a set of machines *M={1,…,m}*. A job $i \in I$ consists of a sequence of operations, $O^i = \{O_{i1},...,O_{ij}\}$, which must be processed consecutively in a fixed order. The parameters are listed as follows.

| | |
|---|---|
| $n$ | total number of jobs |
| $i$ | indices of jobs |
| $s$ | total number of operations, we also call it stage |
| $J_i$ | total number of operations of job $i$ |
| $j$ | indices of operations |
| $m$ | total number of machines |
| $k$ | indices of machines |
| $l$ | sequence of assigned operation on machine |
| $r_i$ | release time of job $i$ |
| $p_{ij}$ | processing time of operation $O_{ij}$ |

## IV. ANALYTICAL PROPERTY

### A. Representation of a Solution

There are $m$ machines in FJSP. The set of these machines is defined as *M={1,…,m}*. The total number of operations is $d = \sum_i J_i$. Number these *d* operations by {1,2,3,...,d}, and define the set *O={1,2,3,...,d}*. The starting time and the release time of operation is denoted as *t(o)* and *r(o)*, respectively. Before we describe the analytical property of the problem, the representation of the solution is given. Lemma 1 shows the proper way to describe one feasible solution.

**Lemma 1.** *In a FJSP with parallel machines and d operations in total, any active schedule can be represented by a 2d dimensional vector,*

$$\vec{Z} = (x_1,...,x_d,y_1,...,y_d)$$

*where* $x_o \in M$ *and* $y_o \in O \cdot x_o$ *is the machine which processes operation o and* $y_o$ *is the operation started in the oth order.*

***Proof:*** Integer array $(x_1,...,x_d)$ chooses a machine for each of the d operations. By given $(x_1,...,x_d)$, the machine choice of each operation is determined. This will transform our FJSP to a JSP. A disjunctive graph can represent a JSP problem. For this JSP, an active schedule is represented by a set of disjunctive arcs, which describes the starting time priority for all operations on each machine. Since integer array $(y_1,...,y_d)$ is the starting time priority sequence for all the operations, so it gives a schedule of the JSP. To sum up, $(x_1,...,x_d,y_1,...,y_d)$ can describe all possible machine assignment strategy and all active schedules for FJSP.

### B. Analytical Property

Since we use a *2d* dimensional integer valued array to describe a solution, the solution space is *2d* dimensional. In this subsection, we provide an important theorem for this problem, with which the dimension of solution space can be reduced to *d*.

For a given starting time priority sequence $(y_1,...,y_d)$, the start time of operation $y_o$ is denoted as $t(y_o)$ and the earliest start time of operation $y_o$ is denoted as $r(t_o) = \max\{r(y_o), t(y_{o-1})\}$. We define an iterative greedy strategy (IGS) as follows.

- Start operation $y_1$ as soon as possible, $t(y_1) = r(y_1)$. Choose the machine for operation $y_1$ arbitrarily.
- Given the machine choice and start time for operation $y_o$, start operation $y_{h+1}$ as soon as possible. That is, $t(y_{h+1}) \geq r(y_{h+1})$, and start operation $y_{h+1}$ when there are idle machines for it. Chose machine arbitrarily.
- Set *h=h+1*, go to step 2. If *h=d*, the solution is obtained.

Based on IGS, Theorem 1 is given.

**Theorem 1.** For a given priority sequence $(y_1,...,y_d)$, IGS which start operation as early as possible will contain an optimal solution to minimize $F_{\max}$.

**Proof:** *The proof is shown as follows.*

**Case 1**. For operation $y_1$, it's obvious optimal to start it as early as possible. And since the idle machines are homogeneous, machine choice could be arbitrary.

**Case 2**. If operation $(y_1,...,y_{h-1})$ are already scheduled. Define time point

$$t_{y_h} = \max\{r(y_h), t(idle\ machine\ exist\ for\ y_h)\}$$

According to Theorem 1, we should start operation $y_h$ at $t_{y_h}$ on an idle machine $m_i$, that is, let $t(y_h) = t_{y_h}$ and $x(y_h) = m_i$. Define a schedule *S* in which we start operation $y_h$ at $t_{y_h} + \Delta t$ on machine $m_j$ and the schedule for the follow-up operations $(y_{h+1},...,y_d)$ is denoted as *S1*. Assume *S* is optimal. To draw a contradiction, we are going to show that if we set $t(y_h) = t_{y_h}$, we will get a better schedule. The proof can be shown as in Fig.1. If in schedule *S* with a objective $F_{\max}$, we didn't use the first idle machine. At time $t_{y_h} + \Delta t$, both $m_j$ and $m_i$ are idle. Define schedule *S'* in which $y_h$ is processed on machine $m_j$, and follow-up schedule *S1* is adjusted by exchange the machine of $m_i$ and $m_j$. Since the homogeneity of machine $m_i$ and $m_j$, $F'_{\max}$ is the same good

as $F_{\max}$. Then we can generate schedule *S''* by moving the start time of operation $y_h$ on $m_j$ forward to $t_{y_h}$. Thus, $F''_{\max} \le F'_{\max} = F_{\max}$. In conclusion, given the priority sequence and the schedule for operation $(y_1,...,y_{h-1})$. We can obtain an optimal solution by IGS. Thus, Theorem 1 is proved.

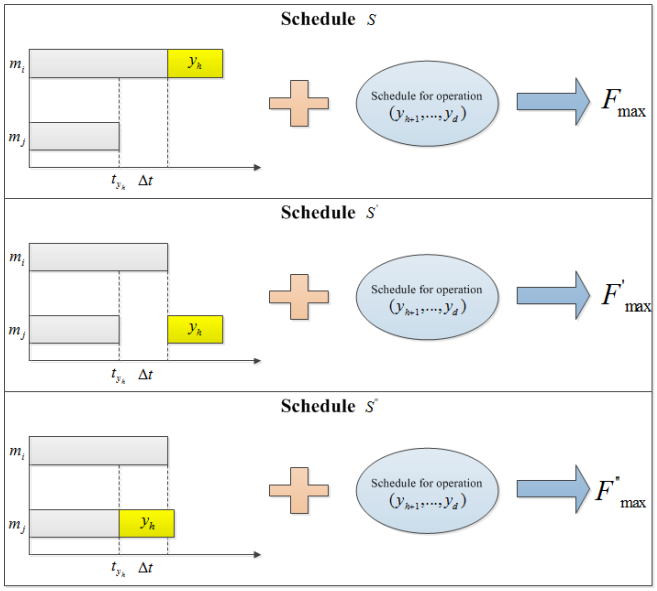


FIGURE I. AN ILLUSTRATION OF PROOF

According to Theorem 1 for a given priority sequence $(y_1,...,y_d)$, we can naturally find one optimal solution by IGS. So we do not need to apply a *2d* dimensional vector to represent the solution in our optimization algorithms. A *d* dimensional vector $(y_1,...,y_d)$ which transverse all the feasible priority sequence is sufficient. By applying Theorem 1, we greatly reduce the solution space we need to search, so the following up algorithms based on this fact will be much more effective.

## V. BRANCH-AND-BOUND ALGORITHM

As mentioned above a feasible priority sequence needs to satisfy the operation precedence sequence of each job. Here we provide a method to use a *d* dimensional vector $(y_1,...,y_d)$ without any constraint to describe a feasible priority sequence. Consider a FJSP with three jobs, three machine groups. Operation $O_{ij}$ is the operation of job *i* which should be processed on machine group *j*. Let $O^j$ denotes the operation set of job *i*, in which the operations are listed according to precedence order. In this case, we have

$$O^1=\{O_{11},O_{12},O_{13}\},\ O^2=\{O_{22},O_{23}\},\ O^3=\{O_{32},O_{31}\}$$

This parallel machine FJSP could be showed in Fig. 2.

There are 7 operations in this example, hence the priority sequence $(y_1,...,y_d)$ will include 7 elements. We construct three stack S1, S2 and S3. S1, S$ and S3 store operation $O_{11},O_{12},O_{13},O_{22},O_{23},O_{32},O_{31}$ from top to bottom, respectively. We define one pop stack sequence (1,1,1,2,2,3,3). This sequence means stack S1 pops three times, then S2 pops two times, and finally S3 pops two times. Popping stack sequence (1,1,1,2,2,3,3) will generate one feasible priority order, which is showed as below.

$$(1,1,1,2,2,3,3)\rightarrow(O_{11},O_{12},O_{13},O_{22},O_{23},O_{32},O_{31})$$

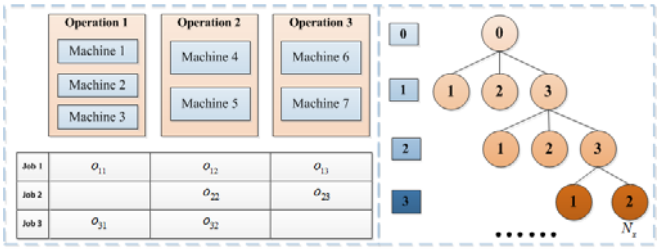


FIGURE II. FJSP WITH THREE JOBS

If we transverse all the different sequence formulated by {1,1,1,2,2,3,3}, we can transverse all the feasible priority order. Put it in another way, one feasible solution is described by a popping stack sequence like (1,1,1,2,2,3,3).

Suppose that our problem has a popping stack array A (in the previous example A={1,1,1,2,2,3,3}). Branch-and-bound algorithm generates a branch tree to enumerate all the different permutation of the stack sequence A. The branch tree of stack sequence {1,1,1,2,2,3,3} is showed in Fig. 2.

Note $N_x$ means the first three popped stack is S3, S3, and S2. So the first three elements in operation priority sequence is $(O_{32},O_{31},O_{22})$. The other 4 elements in the priority sequence is not decided yet. According to Theorem 1, the schedule for $(O_{32},O_{31},O_{22})$ is naturally decided. Then we need to find the lower bound of $F_{\max}$ for node $N_x$. For the partially decided schedule, let $C'_i$ denote the completion time of job *i'* operations in the partially decided schedule. Let $P_{rest}(i)$ denote the sum of processing time of the undecided operations of job *i*. Job *i'* s release time is $r_i$, then the lower bound of $F_{\max}$ is,

$$LB(F_{\max})=\max_i\{C'_i+P_{rest}(i)-r_i\}$$

And the other steps follows the standard branch-and-bound method.

## VI. NUMERICAL EXPERIMENTS

For each test instance, a total of *n* jobs, *m* machines and *s* stages are generated. The release time is generated as uniformly distributed random numbers on [0,T], where T is the horizon time of the release time. The operation number for each job is generated as uniformly distributed random numbers [2,s]. The operation set is randomly generated and the processing time is generated according to the equation 5+10*N(0,1). Finally, all

programs are coded in JAVA and run on a computer with a CPU of 2.0 GHz and an RAM of size equal to 1.0 GB.

We compare the computational time between CPLEX and the proposed branch-and-bound. We evaluate the performance of the proposed branch-and-bound by generating test cases characterized by the value of job numbers ({5,8,10}), the value of machine numbers ({4,6,8,10}), the value of stage numbers ({2,3,4,5}) and the value of horizon time (T=10). Thus, with the values given, a total of 12 test cases can be generated and each case are run 100 times. The summary of these test cases is shown in Table 1. In addition, the objective value of CPLEX and the proposed branch-and-bound is same, thus, we do not display here.

TABLE I. COMPARISON BETWEEN CPLEX AND BRANCH-AND-BOUND

| **Parameters** | | | | **Runtimes(ms)** | |
|---|---|---|---|---|---|
| n | m | s | T | CPLEX | B&B |
| 5 | 4 | 2 | 10 | 334 | 26 |
| 5 | 6 | 3 | 10 | 381 | 9 |
| 5 | 8 | 4 | 10 | 382 | 9 |
| 5 | 10 | 5 | 10 | 598 | 12 |
| 8 | 4 | 2 | 10 | 4761 | 157 |
| 8 | 6 | 3 | 10 | 15412 | 1872 |
| 8 | 8 | 4 | 10 | 2467 | 47 |
| 8 | 10 | 5 | 10 | 13573 | 93 |
| 10 | 4 | 2 | 10 | 318210 | 98713 |
| 10 | 6 | 3 | 10 | 33208 | 1077 |
| 10 | 8 | 4 | 10 | 19573 | 8910 |
| 10 | 10 | 5 | 10 | 15829 | 6211 |

From Table 1, we can find that the computational time of the proposed branch-and-bound is always smaller than that of CPLEX and results can show that the proposed branch-and-bound algorithm has a good performance on solving small-scale problem.

## VII. BRANCH-AND-BOUND ALGORITHM

This paper studies a FJSP with release time and the objective is to minimize the maximum flow time contrary to previous papers. Any active feasible solution can be represented by a *2d* dimensional vector in a FJSP where *d* is the total number of operations. According the specific characteristics of the problem, this paper finds a property which can reduce the *2d* dimension of solution space to *d* dimension. Then several algorithms are proposed to solve different scales of the problem. To solve the problem, a branch-and-bound algorithm is proposed. Finally, through computational experiments, the algorithms are proved to be both effective and efficient in solving the FJSP.

This study can be extended in several directions. A possible extension would be the use of other heuristics in the scheduling algorithm based on the proposed property. In future works, an advanced planning and scheduling which considers the capacity and material constraints can be developed.

## REFERENCES

[1] Balin, S. Non-identical parallel machine scheduling using genetic algorithm. Expert Systems with Applications, 2011, 38(6), 6814-6821.

[2] Baykasoglu, A. Linguistic-based meta-heuristic optimization model for flexible job shop scheduling. International Journal of Production Research,2002, 40(17), 4523-4543.

[3] Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. Computing, 1990, 45(4), 369-375.

[4] Chan, F. T. S., Wong, T. C., & Chan, L. Y. Flexible job-shop scheduling problem under resource constraints. International Journal of Production Research, 2006, 44(11), 2071-2089.

[5] Chan, F. T. S., Choy, K. L., & Bibhushan. A genetic algorithm-based scheduler for multiproduct parallel machine sheet metal job shop. Expert Systems with Applications, 2011, 38(7), 8703-8715.

[6] Chen, J. C., Chen, K. H., Wu, J. J., & Chen, C. W. A study of flexible job shop scheduling problem with parallel machine and reentrant process. International Journal of Advanced Manufacturing Technology, 2008, 39, 344-354.

[7] Choi, I. C., & Choi, D. S. A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent setups. Computers and Industrial Engineering, 2002, 42(1), 43-58.

[8] Kim, Y. K., Park, K., & Ko, J. A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling. Computers and Operations Research. 2003, 30(8), 1151-1171.

[9] Liu, M., Sun, Z. J., Yan, J. W., & Kang, J. S. An adaptive annealing genetic algorithm for the job-shop planning and scheduling problem. Expert Systems with Applications. 2011, 38(8), 9248-9255.

[10] Mastrolilli, M., & Gambardella, L. M. Effective neighbourhood functions for the flexible job shop problem. Journal of Scheduling. 2000, 3(1), 3-20.

[11] Moradi, E., Ghomi, S. M. T. F., & Zandieh, M.. Bi-objective optimization research on integrated fixed time interval preventive maintenance and production for scheduling flexible job-shop problem. Expert Systems with Applications. 2011, 38(6), 7169-7178

[12] Norman, B. A., & Bean, J. C. A genetic algorithm methodology for complex scheduling problems. Naval Research Logistics. 1999, 46(2), 199-211.