AN APPLICATION OF THE GENETIC ALGORITHM TO A TWO-MACHINE ROBOTIC FLOW-SHOP SCHEDULING PROBLEM

Yusuke Hata, Yoshiyuki Karuno and Hirosi Kise
Department of Mechanical and System Engineering
Kyoto Institute of Technology
Sakyo, Kyoto 606-8585, Japan

Abstract

In this paper, we deal with a two-machine robotic flow-shop. There is an intermediate station with a finite capacity bound between the two machines for intermediate operations such as washing, cooling, chip disposal, and so on. Each job is processed on the first machine, and then on the bounded intermediate station, and finally on the second machine. In this paper, we propose a heuristic algorithm to the scheduling problem of minimizing the makespan. The heuristic is designed to handle non-permutation schedules. The performance is examined by means of numerical experiments.

Keywords: Two-machine flow-shop, intermediate operations, non-permutation schedules, genetic algorithm.

1. INTRODUCTION

In this paper, we consider the scheduling problem of minimizing the makespan for a two-machine robotic flow-shop. This flow-shop contains an intermediate station with a finite capacity bound \( Q (1 \leq Q < \infty ) \) between the two machines for some intermediate operations such as washing, cooling, chip disposal, and so on. The intermediate station can process at most \( Q \) jobs at a time. Each of \( n \) jobs is processed on the first machine, and then on the bounded intermediate station, and finally on the second machine. No preemption is allowed. Two robots transport the jobs between the machines and the intermediate station. Such a flow-shop is a typical small-scale FMS (flexible manufacturing system), and actually many cutting machines require such intermediate operations before or after cutting operations.

In a general schedule to the two-machine problem to be discussed here, the processing orderings of jobs on the two machines are allowed to be different. Such a schedule is called a non-permutation schedule. On the other hand, a schedule is called a permutation schedule if the two machines process the jobs in the same ordering. If only permutation schedules are allowed, the problem is referred to as the permutation version. It has been known that the permutation version can be solved in \( O(n^2) \) time if the intermediate station is unbounded (i.e., \( Q = \infty \)) (see (Kise et al. 2000), (Karuno et al. 2002)). However, the original version of the problem where non-permutation schedules are allowed is strongly NP-hard even if \( Q = \infty \) (see (Dell’Amico 1996), (Kise et al. 2000)). Furthermore, even the permutation version is NP-hard for any fixed \( Q \) of the bounded intermediate station, since the problem contains the parallel-machine problem as a special case (e.g., (Hata et al. 2004), (Pinedo 1995)).

In this paper, we propose a heuristic algorithm to the original version of the problem. We use some genetic operators such as crossover and selection (e.g., (Reeves 1997)) to design the heuristic, which delivers non-permutation schedules. For using those genetic operators, a schedule is encoded by a 1-dimensional array with length \( 2n \), where \( n \) is the number of jobs. The processing ordering of jobs on the first machine is represented by the first-half \( n \) elements in the array, and that on the second machine by the latter-half \( n \) elements. For \( 1 \leq k \leq n \), the \( k \)-th element in the array records the \( k \)-th job to be processed on the first machine. On the other hand, for \( 1 \leq k \leq n \), the \( (n+k) \)-th element records an integer \( i \in [1, Q] \), since given a processing ordering of jobs on the first machine, there are at most \( Q \) candidates for the \( k \)-th job to be processed on the second machine. We sort the jobs (i.e., at most \( Q \) candidates) in a non-decreasing order of their arrival times in the intermediate station, and then choose the \( i \)-th job among them as the \( k \)-th job on the second machine. By this encoding, we can apply the traditional crossover operators used for some sequencing problems such as order crossover and free list crossover.

This paper is organized as follows. Section 2 explains the dynamics of the two-machine robotic flow-shop, and defines the mathematical notations of the scheduling problem. Section 3 proposes a heuristic algorithm using some genetic operators. Section 4 examines the performance of the proposed heuristic by means of numerical experiments, and reports the results. Finally, Section 5 gives some concluding remarks.

2. PROBLEM DESCRIPTION

The robotic flow-shop consists of two machines \( M_1 \) and \( M_2 \), an intermediate station IS, two transportation robots \( TR_1 \) and \( TR_2 \), a loading robot \( R_1 \), an unloading robot \( R_2 \), an Input AS/RS and an Output AS/RS. The system layout is
depicted in Figure 1.

![Diagram of a two-machine robotic flow-shop](image)

**Fig. 1 Two-machine robotic flow-shop**

### 2.1 Dynamics of the Robotic Flow-shop

Each of $M_1$ and $M_2$ can process at most one job at a time. The capacity of IS is bounded by an integer $Q (1 \leq Q < \infty)$, i.e., it can process at most $Q$ jobs at a time. Each of $TR_1$ and $TR_2$ can transport at most one job at a time. Each of $R_1$ and $R_2$ can also handle at most one job at a time. Both of Input AS/RS and Output AS/RS have sufficiently large storage for jobs.

The following assumptions are made for the system:

A1) At time zero, there are $n$ jobs available in Input AS/RS.

A2) $R_1$ loads a new job from Input AS/RS to $M_1$ immediately when $M_1$ becomes empty.

A3) $M_1$ starts processing a job immediately after its loading by $R_1$ has finished.

A4) A job whose processing on $M_1$ has finished is unloaded by $TR_1$ if $TR_1$ is available; otherwise the job waits on $M_1$ until $TR_1$ comes back to $M_1$.

A5) After unloading a job from $M_1$, $TR_1$ transports it to IS. Then $TR_1$ drops the job there if IS could accept it; otherwise (i.e., if IS holds $Q$ jobs at that time), the job waits on $TR_1$ until IS could accept it. After dropping the job, $TR_1$ returns back empty to $M_1$.

A6) IS starts processing a job immediately when the job has arrived in IS.

A7) $TR_2$ starts transporting a job from IS if the processing of the job on IS has finished; otherwise $TR_2$ waits until it has finished.

A8) $TR_2$ transports a job from IS to $M_2$, waits until $M_2$ becomes empty if necessary, loads the job to $M_2$, and returns back empty to IS.

A9) $M_2$ starts processing a job immediately after its loading by $TR_2$ has finished.

A10) $R_2$ unloads a job to Output AS/RS immediately after its processing on $M_2$ has finished.

For each job, all of the loading, the processing and the unloading times are considered to be non-negligible and job-dependent.

The transportation and the return times of $TR_1$ and $TR_2$ are considered to be non-negligible and job-independent.

### 2.2 Notations

The following are input parameters:

- $J = \{ j | j = 1, 2, \ldots, n \}$ is the set of $n$ jobs;
- $l_j$ = "loading time" for job $j$ on $M_1$, i.e., time required for $R_1$ to load job $j$ into $M_1$;
- $p_j$ = "processing time" for job $j$ on $M_1$;
- $u_j$ = "unloading time" for job $j$ on $M_1$, i.e., time required for $TR_1$ to unload job $j$ from $M_1$;
- $l_j$ = "loading time" for job $j$ on $M_2$, i.e., time required for $TR_2$ to load job $j$ into $M_2$;
- $p_j$ = "processing time" for job $j$ on $M_2$;
- $u_j$ = "unloading time" for job $j$ on $M_2$, i.e., time required for $R_2$ to unload job $j$ from $M_2$;
- $w_j$ = "processing time" for job $j$ on IS;
- $t_1$ = "transportation time" and "return time" required for $TR_1$ to transport a job from $M_1$ to IS and return from IS to $M_1$, respectively;
- $t_2$ = "transportation time" and "return time" required for $TR_2$ to transport a job from IS to $M_2$ and return from $M_2$ to IS, respectively;
- $Q$ = capacity of IS;
- $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$: a processing ordering of the jobs on machine $M_1$, where $\lambda_j \in J$ stands for the $k$th job to be processed on $M_1$;
- $\mu = (\mu_1, \mu_2, \ldots, \mu_n)$: a processing ordering of the jobs on machine $M_2$, where $\mu_j \in J$ stands for the $k$th job to be processed on $M_2$;
- $\pi = (\lambda, \mu)$: a schedule of the jobs.

Note that permutation version of the problem allows only permutation schedules $\pi = (\lambda, \mu)$ such that $\lambda \neq \mu$. However, in a general schedule $\pi = (\lambda, \mu)$, $\lambda \neq \mu$ is allowed. $\pi = (\lambda, \mu)$ determines the following key time instants:

- $S_1[j]$ = the time instant when $R_1$ starts loading job $j$ on $M_1$;
- $F_1[j]$ = the time instant when job $j$ is released from $M_1$ by $TR_1$;
- $S_2[j]$ = the time instant when IS starts processing job $j$ (equivalently, arrival time of job $j$ in IS);
- $C[j]$ = the time instant when IS has finished processing job $j$;
- $F_2[j]$ = the time instant when $TR_2$ starts transporting job $j$ from IS to $M_2$;
- $S_2[j]$ = the time instant when $TR_2$ starts loading job $j$ on $M_2$;
- $F_2[j]$ = the time instant when job $j$ is released from $M_2$ by $R_2$, which is the completion time of job $j$. 

34
\( F_{\text{max}}(\pi) \): the makespan obtained by a schedule \( \pi \), i.e.,
\[
F_{\text{max}}(\pi) = F_2[\mu_n];
\]
\( E_f(\pi) \): the machine efficiency by a schedule \( \pi \), i.e.,
\[
E_f(\pi) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} \left( k[j] + \rho_k[j] + u_k[j] \right)}{2 \cdot F_{\text{max}}(\pi)}.
\] (1)

The objective is to find an optimal schedule \( \pi = \pi^* \) that minimizes the makespan \( F_{\text{max}}(\pi) \), equivalently, that maximizes the machine efficiency \( E_f(\pi) \).

3. HEURISTIC ALGORITHMS

In this section, we propose a heuristic algorithm to the problem. We use some genetic operators such as crossover and selection (e.g., (Reeves 1997)) to design the heuristic, which delivers non-permutation schedules. Since the basic framework of the genetic algorithm is well-known, we focus on some aspects of the application to the problem.

First, we encode a schedule by a 1-dimensional array with length \( 2n \). The processing ordering of jobs on the first machine is represented by the first-half \( n \) elements in the array, and that on the second machine by the latter-half \( n \) elements. For \( 1 \leq k \leq n \), the \( k \)th element in the array records \( \lambda_k \), i.e., the \( k \)th job to be processed on the first machine. On the other hand, for \( 1 \leq k \leq n \), the \( (n+k) \)-th element records an integer \( i \in [1,Q] \). Since the intermediate station IS can hold at most \( Q \) jobs at a time, if a processing ordering \( \lambda \) on the first machine is given, there are at most \( Q \) candidates for \( \mu_k \), i.e., the \( k \)th job to be processed on the second machine. The jobs (i.e., at most \( Q \) candidates) are sorted in a non-decreasing order of their arrival times in the intermediate station IS (note that the arrival times can be computed if \( \lambda \) is given). Thus, if the \( (n+k) \)-th element in the array records a certain integer \( i \in [1,Q] \), it means that the \( k \)th among at most \( Q \) candidates is chosen as the \( k \)th job on the second machine.

An initial population of arrays is produced as follows. For each array in the initial population, the first-half \( n \) elements is given by a randomly produced permutation on \( \{1,2,\ldots,n\} \), and each element in the latter-half is given by a randomly produced integer, using a decreasing step distribution on \( [1,2,\ldots,Q] \), where \( Q \) is defined as \( \min(Q, n-k+1) \) for the \( (n+k) \)-th element \( 1 \leq k \leq n \) (notice that there are \( Q \) candidates for \( \mu_k \)). More precisely, for each element in the latter-half, \( i \in [1,2,\ldots,Q] \) is chosen with probability \( 2(Q-i+1)/(Q(Q+1)) \). We refer to this manner as D-STEP producer. Note that we set all elements in the latter-half to 1 with probability one, the resulting schedule is a permutation schedule. When we use a uniform distribution on \( [1,2,\ldots,Q] \) instead of the D-STEP (i.e., any \( i \in [1,2,\ldots,Q] \) is chosen with probability \( 1/Q \)), we call it UNIFORM producer.

Next, we describe how to yield a child array from two parent arrays. For the first-half of the parents, we use the uniformly order crossover, denoted by \( \text{OX(U)} \), or the uniformly free list crossover, denoted by \( \text{FLX(U)} \) (e.g., see (Yagiura and Ibaraki 1994)). For the latter-half, we consider the following crossover operator (in order to facilitate the procedure, we present the crossover together with the encoding steps):

\[ \lambda\text{-DX(U)} \] (uniformly \( \lambda \)-dependent crossover)

**Step 1 (Initialization)**
- For each of two parent schedules \( \pi^k \) for \( k = 1,2 \), \( j_{h}^{k} = \lambda_{h}^{k} \) for \( h = 1,2,\ldots,n \); \( R_{\pi^k} := \{ j_1^{k}, j_2^{k}, \ldots, j_{n}^{k} \} \); \( i := 1 \); Go to Step 2.

**Step 2 (Ending test of encoding)**
- If \( i > n \), \( \pi^{\lambda} := \{ \mu_1^{\lambda}, \mu_2^{\lambda}, \ldots, \mu_n^{\lambda} \} \) for \( k = 1,2 \), and go to Step 4; otherwise, go to Step 3.

**Step 3 (Encoding)**
1. Find \( l \) such that \( j_{l}^{k} = \mu_{i}^{\lambda} \);
2. \( \pi^{\lambda} := l \); \( R_{\pi^\lambda} := R_{\pi^k} \cup \{ j_{l}^{k} \} \); \( j_{h}^{k} := j_{h+1}^{k} \) for \( k = 1,2 \) and \( h = l+1,\ldots,n-l \);
3. \( i = i+1 \); Return to Step 2.

**Step 4 (Crossover)**
1. Create \( n \)-bits 0-1 mask \( m \in \{0,1\}^n \);
2. If \( m[i] = 1 \), \( \tilde{\mu}_i = \mu_i^{\lambda} \), otherwise \( \tilde{\mu}_i = \mu_i^{\lambda} \) for \( i = 1,2,\ldots,n \);
3. Output a child schedule with \( \mu \) decoded from \( \tilde{\mu} = \{ \tilde{\mu}_1, \tilde{\mu}_2, \ldots, \tilde{\mu}_n \} \) by the opposite way of Step 3.

We remark that if two parent schedules are feasible with respect to the capacity constraint of the intermediate station, the crossover \( \lambda\text{-DX(U)} \) yields a feasible child schedule. An example of \( \lambda\text{-DX(U)} \) is illustrated in Figure 2, where we assume that \( \lambda = \lambda^1 = \lambda^2 \) for \( n^1 = (3,1,2,4,5) \) and \( n^2 = (\lambda^1, \lambda^2) \). By replacing the roles of \( \mu^1 \) and \( \mu^2 \) each other, we can obtain another child.

\[
\lambda^1 = (3,1,2,4,5)
\]
\[
\mu^1 = P1 3 2 4 1 5 \rightarrow \mu^2 = P1 1 2 2 1 1
\]
\[
P2 1 2 3 5 4 \rightarrow P2 2 2 1 2 1
\]
\[
\mask = 0 1 1 0 0
\]
\[
C1 1 2 4 5 3 \rightarrow C1 2 2 2 2 1
\]

**Fig. 2 An Example of \( \lambda\text{-DX(U)} \)**

4. NUMERICAL EXPERIMENTS

In this section, we denote by \( G1 \) the genetic heuristic with \( \text{OX(U)} + \lambda\text{-DX(U)} \) and D-STEP producer, by \( G2 \) the heuristic with \( \text{FLX(U)} + \lambda\text{-DX(U)} \) and D-STEP producer, and by \( G3 \) the heuristic with \( \text{OX(U)} + \lambda\text{-DX(U)} \) and UNIFORM producer. For all of the heuristics \( G1, G2 \) and \( G3 \), we set the
number of generations to 100 and also the size of population to 100. The tournament/elite strategy is adopted as the selection, where the number of elites is set to be six. Furthermore, all the heuristics do not use any mutation.

We also give the results of the $O(n^2)$ time algorithm proposed by (Kise et al. 2000), denoted by OP, which delivers an optimal permutation schedule for the case of $Q = \infty$.

The problem instances to be tested are randomly generated as follows:

- The total number of jobs: $n = 100$.
- The rate of the number of jobs with intermediate operations to the total number of jobs: $\alpha = 50\%$.
- The loading and unloading times: $l_i(j), m_i(j), e_i(j)$ and $u_r(j)$ are uniformly random integers in $[5, 15]$.
- The processing times on $M_1$ and $M_2$: $p_1[j]$ and $p_2[j]$ are uniformly random integers in $[13, 187]$.
- The processing times on IS: $w[j]$ are uniformly random integers in $[13, 187]$.
- The transportation and return times: $t_1 = t'_1 = t_2 = t'_2 = 28$.

The program is written in C. It is compiled by Microsoft Visual C++, and run on a personal computer EPSON DIRECT Endeavor/Pro-720L. When the number of generations is 100 (and the size of population is also 100), the CPU time of the proposed heuristic G1 is at most 4[sec] for a problem instance with 100 jobs. In all of Figures 3, 4 and 5, each data indicates the mean value for 50 problem instances tested.

![Fig. 3 Comparison between crossover operators OX(U) and FLX(U)](image)

Figure 3 provides the machine efficiency obtained by G1 and that by G2, by which we compare the crossover operator OX(U) with FLX(U), while Figure 4 shows the machine efficiency obtained by G3, by which we compare the initial population producer D-STEP with UNIFORM. From these figures, we can see that G1 yields better schedules than G2 and G3. For a job $u_k$, when the corresponding element in the latter-half of the 1-dimensional array has a relatively large $i \in [1, 2, \ldots, Q]$, it tends to stay in the intermediate station for a long time. In particular, we often observe such jobs in the schedules obtained by G3.

![Fig. 4 Comparison between initial population producers D-STEP and UNIFORM](image)

In Figure 5, we compare the proposed heuristic G1 with the OP. The G1 yields better schedules than the OP in the range of $Q < 15$, and the machine efficiency in this range is as high as that obtained by the OP when $Q = 20$. However, the OP is better in the range of $Q > 15$. It should be mentioned that the results obtained by the OP when $Q \geq 18$ were the same machine efficiency as that when $Q = 100 (= n)$.

![Fig. 5 Comparison with the optimal permutation schedule for the case of $Q = \infty$](image)

5. CONCLUDING REMARKS

We considered the scheduling problem of minimizing the makespan, equivalently maximizing the machine efficiency for a two-machine robotic flow-shop with a bounded intermediate station. The capacity bound is denoted by $Q (\leq \infty)$. When the processing orderings of jobs on the two machines are allowed to be different (i.e., non-permutation schedules), the problem is strongly NP-hard even if the intermediate station is unbounded. In this paper, we proposed a heuristic al-
algorithm to the problem, which was designed to handle non-
permutation schedules using some genetic operators such as
crossover and selection.

The natural representation of schedules is not necessarily
the most effective for the genetic operators. We encoded a
schedule by a 1-dimensional array with length 2n, where n
is the number of jobs. The processing ordering on the first
machine is represented by the first-half n elements in the ar-
ray, while each element in the latter-half of the array records
an integer in [1, Q] to represent the processing ordering on
the second machine. The proposed heuristic includes the
encoding scheme so that the traditional crossover operators
yield a feasible child schedule with respect to the capacity
constraint of the intermediate station if two parent schedules
are feasible.

Numerical results showed that for the case of smaller ca-
city bounds, the machine efficiency obtained by the pro-
posed heuristic was as high as that by the optimal permuta-
tion algorithm for the problem with an unbounded inter-
mediate station. In the future research, however, we should
analyze the proposed heuristic more carefully with various
conditions of numerical experiments to examine the effec-
tiveness. Also for the case of larger capacity bound, the per-
formance of the proposed heuristic may be improved by us-
ning some different distribution to choose integers from [1, Q]
in the producer of the initial population.

Acknowledgement

This research was partially supported by a Scientific
Grant in Aid from the Ministry of Education, Culture,
Sports, Science and Technology of Japan.

References

Hata, Y., Y. Karuno and H. Kise (2004). A heuristic algo-

rithm for a two-machine flowshop scheduling prob-
lem with a bounded intermediate station. Transac-
tions of the Japan Society of Mechanical Engineers (in
Japanese), in press.
Karuno, Y., Y. Hata and H. Kise (2002). Empirical analy-

sis of characteristics of a two-machine robotic flow-
shop with intermediate operations. Transactions of the
Japan Society of Mechanical Engineers (in Japanese),
Kise, H., Y. Karuno and J.-L. Cheng (2000). Optimal per-
mutation and non-permutation scheduling for a two-

machine robotic unit with an intermediate station.
Proceedings of the IASTED International Conference,
Systems, pp. 61–92. Prentice-Hall, Englewood Cliffs,
New Jersey.

Reeves, C. R. (1997). Genetic algorithms for the operations
researcher. INFORMS Journal on Computing, Vol. 9,
No. 3, pp. 231–250.
operators for sequencing problems. T. IEE Japan (in