SCHEDULING JOB OPERATIONS IN AN AUTOMATIC ASSEMBLY LINE

Kang G. Shin and Qin Zheng
Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122.

Abstract
This paper addresses the problem of scheduling job operations in an automatic assembly line used for manufacturing a small to medium volume of mixed workparts. The assembly line model used here differs from the classical flow shop model in the following three aspects: (1) there are no buffers at machine stations, (2) constraints associated with the material transport system are included, and (3) for each batch of production, workparts are distinguished in groups, rather than individually. An optimal algorithm that requires very little computation is derived first by minimizing the total finish time for two machine assembly lines. This result is then generalized to the problem of scheduling an assembly line with \( m > 2 \) machines processing single-operation jobs. In order to reduce the computational complexity of the latter problem, heuristic algorithms are also proposed and shown to work quite well for all the cases considered. Finally, discussed is the solution to the problem of scheduling an assembly line with \( m > 2 \) machines processing multi-operation jobs.

1 Introduction

In an automatic assembly line, a sequence of operations are performed by machine stations, and workparts are automatically transferred from one station to the next station. Usually, an automatic assembly line is designed for mass production of some commodity, such as home appliances and cars. Contemporary research has mainly focused on the problem of allocating work to different stations such that all stations have nearly identical completion time [5]. Since the station with the maximum completion time dictates the assembly cycle, a perfectly balanced assembly line with a given number of stations provides the highest production rate.

In recent years, flexible manufacturing systems (FMS) have been drawing considerable attention from both research and commercial communities. The basic idea of FMS is to make assembly lines capable of simultaneously manufacturing mixed workparts, and efficiently handling a small to medium volume of workparts, thus allowing for rapid responses to market changes and use of expensive equipment for multiple purposes. If an assembly line is designed to be capable of manufacturing a small to medium volume of mixed workparts, the problem of scheduling operations on the different machine stations of this assembly line would become a key to the issue of productivity. Hundreds of robots, sensors, and millions of dollars' worth of computer-controlled equipment would not be cost-effective if they are under-utilized or spend most of the time working on wrong parts because of poor planning [6].

Various scheduling problems for classical flow shop models have long been studied by numerous researchers [1, 3]. However, most of these flow shop problems are known to be \( \text{NP-hard} \) for which it is impossible to obtain optimal solutions with reasonable computational efforts except for those of very small size [4]. Even those problems for which optimal polynornial time or good heuristic algorithms have been found cannot be directly applied to the assembly line scheduling problem for the following reasons:

- In a flow shop model, it is assumed that there is no restriction on buffer sizes at machine stations, which is not the case for automatic assembly lines.
- Material transport systems are not figured in the flow shop models.
- Workparts are distinguished individually in the flow shop model, whereas it is more suitable to distinguish them in groups for an assembly line.

In this paper, we propose a model for automatic assembly lines which can be used for manufacturing a small to medium volume of mixed workparts. First, we shall derive an optimal scheduling algorithm for two machine assembly lines by minimizing the total finish time. Based on this result, optimal and heuristic algorithms for assembly lines each with more than two machines are then developed and analyzed.

C. Srisundarajah et al [8] investigated a similar problem of scheduling a production line with a circular conveyor. However, they restricted all machines' processing times to be identical, and hence, their solution algorithm is optimal only if the job processing time is equal to one unit of time, thus limiting its applicability. There are also some other papers concerning the problem of scheduling operations in assembly lines. For example, Burns et al [2] derived some analytical principles to reduce the set-up cost and increase the capacity utilization, rather than minimizing operation times, of assembly lines. O'Connell et al [7] simulated some simple heuristic scheduling algorithms for a flexible transfer line.

This paper is organized as follows. The structure of the automatic assembly line to be considered is described and the scheduling problem is formally stated in Section 2. Necessary terms are
also defined there. In Section 3, the optimal solutions for two machine assembly lines are derived on the basis of which heuristic algorithms for $m > 2$ machine assembly lines are developed and evaluated in Section 4. The paper concludes with Section 5.

2 Terminology and Problem Formulation

As shown in Fig. 1, the automatic assembly line under consideration is composed of a linear conveyor, $m \geq 1$ machines, $M_1, M_2, \ldots, M_m$, a job entrance and a job exit. There are $T_i$ pallets on the conveyor, each of which can carry one job (or part to be worked on) at a time. The conveyor moves forward one pallet every unit of time. Although a machine may be capable of executing more than one kind of job operations, in order to reduce the set-up cost, we assume that each machine is allowed to execute only one kind of operations during one batch of production. For $1 \leq i \leq m$ let $T_i$ denote the processing time of a job by machine $M_i$.

![Fig. 1. Configuration of an automatic assembly line.](image)

Jobs are distinguished in groups. Each group is denoted by $J_{i_1, \ldots, i_k}$ with $N_{i_1, \ldots, i_k}$ identical jobs, where the subscript represents the order of machines to process that type of jobs, $1 \leq i_1, \ldots, i_k \leq m$, $k \geq 1$. For example, each job of type $J_{12}$ must be processed sequentially by $M_1, M_3$ and $M_2$.

The automatic assembly line runs jobs according to the following rules:

1. Jobs enter and leave the assembly line only through the job entrance and the job exit, respectively. Only one job can enter and one job can leave the assembly line during each unit of time because a pallet can carry only one job.

2. Jobs enter and leave a machine through a gate connecting the machine to the conveyor. In one unit of time, a machine can output a finished job to a pallet at the gate and accept a job from the same pallet.

3. There are no buffers at machine gates. So, when a machine is busy, it cannot accept another job, and when the pallet at the machine gate is occupied by a job which does not need immediate processing by this machine, the machine cannot output a job to the pallet.

4. Every job must leave the system when it arrives at the job exit. If a job has not yet been completed when it arrives at the job exit, it will be transferred back to the job entrance in $T_i$ units of time and re-enter the assembly line later.

Under the above rules, our assembly line scheduling problem can be stated as follows: Choose an input sequence of jobs to minimize the total finish time, which is defined as the time period between the input of the first job and the input of the last job.

For the convenience of presentation, it is necessary to introduce the following terms.

**Definition 1**: A machine is said to be free if it is not processing any job. This includes two cases: 1) there is no job being processed by the machine, and 2) there is a finished job in the machine that is waiting to be outputed to the conveyor. In either case, the machine is ready to accept a new job. (By the second rule mentioned above, a new job can be accepted in the latter case.)

**Definition 2**: A job is said to match a machine if the first remaining operation of the job is required to be processed by this machine. For example, jobs of type $J_{12}$ match only $M_1$ but weakly match both $M_1$ and $M_2$. Note that a $J_{12}$-type job will become a $J_{13}$-type job after it is processed by $M_2$.

**Definition 3**: Let $R(t)$ denote the pallet at the job entrance at time $t$. $P_0(t)$ is said to match a machine if (i) the machine is free at the time $P_0(t)$ reaches the machine, and (ii) there is at least one job which weakly matches the machine will be inputed to the assembly line. $P_0(t)$ is said to weakly match a machine if only (i) is satisfied.

**Definition 4**: A job is said to be missed at time $t$ if (i) $P(t)$ matches the machine, and (ii) no job is inputed to the machine at the time $P(t)$ reaches the machine. A collision is said to occur at time $t$ if $P(t)$ matches more than one machine. Time $t$ is said to be a collision time without machine miss if no machine is missed at $t$. For instance, if $P(t)$ matches $M_1$ and $M_2$, then $t$ can be made a collision time without machine miss by inputing a $J_{12}$-type job at time $t - T_1$ and a $J_1$-type job at $t$.

**Definition 5**: An input sequence is said to be feasible if it can have all the jobs processed in a finite time. A feasible sequence is denoted by $z = (x(1), x(2), \ldots)$, where $x(t)$ represents the type of job inputed at time $t$. $x(t) = 0$ if no job is inputed at time $t$. Let $T(z)$ denote the total finish time for an input sequence $z$ and $G$ denote the set of all feasible input sequences.

**Definition 6**: A job type is said to be available at time $t$ if there is at least one job of this type waiting to be inputed at time $t$. Otherwise, this job type is said to be unavailable. A job type is said to be in short supply at time $t$ if the input rules require a job of this type to be inputed at time $t$, but this job type is unavailable at time $t$.

**Definition 7**: Let $F, G$ be two subsets of $\Omega$. $F$ is said to dominate $G$ if $\forall x \in F$, $\exists x_1 \in F$ such that $T(x_1) \leq T(x)$. The set of optimal input sequences, denoted by $S^*$, is defined as: $S^* = \{z : z \in \Omega, T(z) \leq T(z_1), \forall z_1 \in \Omega\}$.

3 A Special Case: Two Machine Assembly Lines

In this section, the problem of scheduling operations in a two machine assembly line is addressed. The results obtained for this special case will be generalized to $m > 2$ machine assembly lines in the next section.
In a two machine assembly line, there could be three elementary job types: \( J_1, J_2 \) and \( J_3 \). A job of any elementary type can be completed by inputing it just once to the assembly line. Jobs of non-elementary types need to be inputed more than once, and thus, can be viewed as compositions of elementary type jobs. For instance, a \( J_{31} \)-type job must be inputed first as a \( J_2 \)-type job and then as a \( J_1 \)-type job, so it can be treated as a composition of a \( J_2 \)-type job and a \( J_1 \)-type job. So is a \( J_{31} \)-type job: it is a composition of \( J_1 \) and \( J_3 \) job. Since the existence of non-elementary type jobs does not make any difference to our scheduling problem, we shall first derive an optimal algorithm for a two machine (\( M_1, M_2 \))—three job type (\( J_1, J_2, J_3 \)) assembly line scheduling problem. Extensions of this algorithm to those assembly lines with more machines and job types will be addressed later in Section 4.

3.1 A Lower Bound of the Finish Time

A lower bound of the finish time for a two machine (\( M_1, M_2 \))—three job type (\( J_1, J_2, J_3 \)) problem is derived to construct and prove an optimal scheduling algorithm in the next subsection.

Let \( N_{11}, N_{12}, N_{13} \) be respectively the number of jobs of type \( J_1, J_2, J_3 \), and suppose \( N_{11} + N_{12} > 0, N_{12} + N_{13} > 0 \). If the last two inequalities do not hold, the problem would become a one machine scheduling problem which is easier to solve. For any \( x \in \Omega \), let \( n_{1}(x), t_{1} \) denote the total number of times \( M_1 \) is missed during \( [1, t-1] \). Then, the last job matching \( M_1 \) will be inputed to the assembly line at the time

\[
T_w(x) = \min\{T_{w1}(x), T_{w2}(x)\}
\]

Let \( T_w(x) = \min\{T_{w1}(x), T_{w2}(x)\} \). Then, from Definition 3 and Definition 5, \( t \) is a collision time if (1) \( P_k(t) \) weakly matches both machines, and (2) \( t \leq T_w(x) \).

Let \( t_1 = k_1 T_1 + 1 = \ell_2 T_2, t_2 = k_2 T_1 = \ell_2 T_2 + 1 \) and \( t_3 = k_1 T_1 + 1 + \ell_2 T_2 > 0 \), where \( k_1, k_2, \ell_1, \ell_2, \ell_1, \ell_2 \) are the smallest nonnegative integers (or \( \theta \) if \( \theta \) does not satisfy these equations). The physical meaning of \( t_1 \) is that if \( t_0 \) is a collision time, and \( M_1 \) is missed at \( t_0 \), then at least one of the following events will happen in the time period \( [t_0 + 1, t_0 + t_1] \). \( E_1 = \{M_1 \text{ is missed once}\}, E_2 = \{M_2 \text{ is missed once}\}, E_3 = \{A \text{ collision happens without machine miss}\} \). This fact can be verified by observing that if neither \( E_1 \) nor \( E_2 \) happens in \([0 + 1, t_0 + t_1 - 1] \), then \( t_0 + t_1 - 1 \) is a collision time. Thus, at least one of \( E_1, E_2, E_3 \) will happen at \( t_0 + t_1 \). Similarly, the physical meaning of \( t_2 \) is that if \( t_0 \) is a collision time and \( M_2 \) is missed at \( t_0 \), then at least one of \( E_1, E_2, E_3 \) will happen during \([t_0 + 1, t_0 + t_2] \). If \( t_0 \) is a collision time without machine miss, then at least one of \( E_1, E_2, E_3 \) and \( E_4 \) will happen during \([t_0 + 1, t_0 + t_2] \). Roughly speaking, \( t_1(t_2, t_3) \) can be thought as the maximum interval between the occurrence of \( E_1(E_2, E_3) \) and that of one of \( E_1, E_2 \) and \( E_3 \). Thus, none of \( E_1, E_2 \) or \( E_3 \) could happen after time \( n_1(x, T(x)) + n_2(x, T(x)) + n_3(x, T(x)) \), where \( n_1(x, t) \) denotes the total number of times collisions occur without machine miss during \([1, t] \). This concept is developed formally in the following theorem.

Theorem 1. For any \( x \in \Omega \), \( n_1(x, T(x)), n_2(x, T(x)), n_3(x, T(x)) \) must satisfy the following inequalities:

\[
\begin{align*}
(1) & \ n_1(x, T(x)) + n_2(x, T(x) + n_3(x, T(x)) \geq T_w(x). \\
(2) & \ n_1(x, T(x)) + n_2(x, T(x)) + T_w(x) \geq T_1. \\
(3) & \ n_1(x, T(x)) \leq N_{12}.
\end{align*}
\]

For the limitation of space, all proofs are not presented in this paper. Theorem 1 specifies some constraints on the values of \( n_1(x, T(x)), n_2(x, T(x)), n_3(x, T(x)) \). Given \( n_1(x, T(x)), n_2(x, T(x)), n_3(x, T(x)) \), the finish time of \( x \) can be computed as

\[
T(x) = \max\{(N_1 + N_{12} - 1)T_1 + n_1(x, T(x)), (N_2 + N_{12} - 1)T_2 + n_2(x, T(x))\} + 1.
\]

Thus, the solution of the following nonnegative integer programming problem is a lower bound of the finish time:

\[
\min \{\max\{(N_1 + N_{12} - 1)T_1 + n_1, (N_2 + N_{12} - 1)T_2 + n_2\} + 1.
\]

subject to

\[
\begin{align*}
& n_1 + n_2 + n_3 + \leq \min\{(N_1 + N_{12} - 1)T_1 + n_1, (N_2 + N_{12} - 1)T_2 + n_2\} + 1. \\
& n_1 + n_2 + n_3 \geq T_1. \\
& n_1 + n_2 \leq N_{12}.
\end{align*}
\]

This problem can be easily solved by the following algorithm.

Algorithm 1

Step 1. Set \( n_1 = n_2 = n_3 = 0 \). Compute \( T_1, T_2 \) and \( T_3 \) (replace with a large positive integer if not exist).

Step 2. If \( n_1 + n_2 + n_3 \geq \min\{(N_1 + N_{12} - 1)T_1 + n_1, (N_2 + N_{12} - 1)T_2 + n_2\} + 1 \), then stop. The solution is \( (N_1 + N_{12} - 1)T_1 + n_1, (N_2 + N_{12} - 1)T_2 + n_2 \) and \( n_1, n_2, n_3 \) are the minimizers. Otherwise, goto Step 3.

Step 3. If \( n_1 + n_2 + n_3 \geq T_1 \) and \( n_3 < N_{12} \), then let \( n_3 = n_3 + 1 \), goto Step 2. Otherwise, goto Step 4.

Step 4. If \( (N_1 + N_{12} - 1)T_1 + n_1 \leq (N_2 + N_{12} - 1)T_2 + n_2 \), then let \( n_1 = n_1 + 1 \), goto Step 2. Otherwise, let \( n_1 = n_2 + 1 \), goto Step 2.

Since \( T_1 \geq 1 \), this algorithm will terminate in a finite number of steps. Let \( T^* \) denote the solution and \( n_1^*, n_2^*, n_3^* \) denote the minimizers.

3.2 An Optimal Scheduling Algorithm

We propose the following scheduling algorithm for the two machine (\( M_1, M_2 \))—three job type (\( J_1, J_2, J_3 \)) problem.

Algorithm 2

Step 1. Apply Algorithm 1 to determine the minimizer \( n_3^* \).

Step 2. Generate an input sequence \( x \) using the following input rules at time \( t \).
RULE 1: If \( R_0(t) \) does not match any machine, \( z(t) = 0 \).

RULE 2: If \( R_0(t) \) matches only \( M_2 \), \( z(t) = J_2 \).

RULE 3: If \( R_0(t) \) matches only \( M_1 \), then \( z(t) = J_1 \) if \( N_{12}(t) > n_{12}^* \), and \( z(t) = J_1 \) otherwise.

RULE 4: If \( R_5 \) matches both machines, then
\[
\begin{align*}
&\text{If } t > T_1 \text{ and } N_{12}(t) > 0, \text{ then } z(t - T_1) = J_2, \quad n_{12}^* = n_{12}^* - 1, \text{ and } z(t) = J_2 \text{ if } N_{12}(t) > n_{12}^* \text{, and } z(t) = J_1 \text{ otherwise.} \\
&\text{If } t \leq T_1 \text{ or } N_{12}(t) = 0, \text{ then}
- \quad \text{If } (N_{12}(t) + N_2(t) - 1)T_1 < (N_{12}(t) + N_2(t) - 1)T_1, \text{ then } z(t) = J_2. \\
- \quad \text{If } (N_{12}(t) + N_2(t) - 1)T_1 \geq (N_{12}(t) + N_2(t) - 1)T_1, \text{ then } z(t) = J_2 \text{ if } N_{12}(t) > n_{12}^* \text{ and } z(t) = J_1 \text{ otherwise.}
\end{align*}
\]

where \( N_1(t), N_2(t), N_{12}(t) \) are the numbers of jobs of type \( J_1, J_2, J_{12} \) which are not yet inputted at time \( t \) plus those that are inputted before time \( t \) but will leave the conveyor as unfinished jobs.

The optimality of Algorithm 2 is stated formally in the following theorem under the assumption that no job will be in short supply.

Theorem 2. The input sequence \( S^* \) generated by Algorithm 2 belongs to \( S^* \) if no job is in short supply during the entire input process.

Several remarks are worth making on Algorithm 2 as follows.

1. The assumption used in Theorem 2 is that no job is in short supply during the entire input process. For Algorithm 2, a short supply could happen only if \( n_{12}^* < N_{12} \), i.e., some \( J_{12} \)-type jobs need to be inputted twice. Recall that a \( J_{12} \)-type job becomes a \( J_2 \)-type job after being processed by \( M_2 \). If all the \( J_2 \)-type jobs at the job entrance have been inputted before a \( J_2 \)-type job generated from a \( J_{12} \)-type job reaches the job entrance, and the input rules require a \( J_2 \)-type job to be inputted at this time, \( J_2 \)-type jobs will be in short supply. Note that RULE 3 and RULE 4 of Algorithm 2 have made those \( J_{12} \)-type jobs needing to enter twice inputted as early as possible, thus reducing the possibility of short supply. If this still cannot prevent short supply of jobs, we can simply let \( z(t) = 0 \) at the time \( t \), when a short supply occurs and continue using Algorithm 2 to get a feasible input sequence. However, in such a case there is no guarantee that the resulting input sequence is still optimal.

2. If Algorithm 2 is applied to a two machine (\( M_1, M_2 \))—two job type (\( J_1, J_2 \)) scheduling problem, an optimal input sequence will always be generated. The reason is that for the two job type problem, \( N_{12} = n_{12}^* = 0 \), so no job will be in short supply and by Theorem 2, the resulting input sequence is always optimal. Moreover, Step 1 is no longer needed for the two job type problem, thus making Algorithm 2 an online algorithm. No pre-computation is necessary and the algorithm can be readily adapted to changes in the number of jobs to be processed. For example, if \( k \) \( J_1 \)-type jobs arrive at the system at time \( t > 1 \), one can simply update \( N_1(t) = N_1(t) + k \). The adapted algorithm will make the whole batch of jobs processed in minimum time.

3. As stated earlier, a non-elementary job can be viewed as a composition of several elementary jobs. Thus, one can decompose all non-elementary jobs into elementary ones and let \( N_1, N_2, N_{13} \) be the total numbers of \( J_1, J_2, J_{13} \) type jobs, respectively, after such a decomposition. Then, Algorithm 2 can be applied and Theorem 2 still holds. In order to reduce the possibility of short supply of jobs, those jobs needing to cycle more than once should be inputted first. For instance, a \( J_{21} \)-type job needs two cycles and a \( J_{211} \)-type job needs three cycles to complete. If Algorithm 2 happens to yield \( z(t) = J_2 \), then a \( J_{211} \)-type job should be inputted. Recall that inputting jobs with a maximal number of cycles first is a heuristic to reduce the possibility of short supply of jobs.

4 Scheduling \( m > 2 \) Machine Assembly Lines

Scheduling \( m > 2 \) machine assembly lines is naturally more difficult than scheduling \( 2 \) machine assembly lines. In this section, we first find a dominant subset of \( M \) for assembly lines with \( m > 2 \) machines processing single-operation jobs. An enumeration algorithm is then proposed for searching this dominant subset to obtain an optimal solution. To reduce the computational complexity, heuristic algorithms are also developed. Simulation results indicate that these algorithms work quite well. Finally, we shall discuss the general \( m > 2 \) machine multi-operation job scheduling problem for which a heuristic solution is proposed.

4.1 \( m > 2 \) Machine Single-Operation Job Scheduling Problem

A job is said to be of single-operation type if it needs to be processed by one and only one machine. Otherwise, it is said to be of multi-operation type. For the \( m > 2 \) machine single-operation job scheduling problem, there could be at most \( m \) different job types \( J_1, J_2, ..., J_m \). We propose the following input rules for a set \( M(t) \), of machines which \( R_0(t) \) matches at time \( t \).

R1: If \( M(t) = \emptyset \), then \( z(t) = 0 \).

R2: If \( M(t) \neq \emptyset \), then let \( z(t) \) be any job type that matches a machine in \( M(t) \).

Clearly, in case \( |M(t)| > 2 \), R2 cannot determine a unique value for \( z(t) \). Thus, there could be more than one feasible input sequence satisfying R1 and R2. Let \( \Omega_1 \) be the set of all such sequences. Then, we have the following theorem.

Theorem 3. \( \Omega_1 \) dominates \( \Omega \).

Since R1 and R2 may not determine a unique input sequence, the following depth-first search algorithm can be used to find an optimal sequence \( z^* \) in \( \Omega_1 \).

Algorithm 3

Step 1. Set the optimal finish time \( T^* = \infty \), and the current time \( t = 0 \).

Step 2. Let \( t = t + 1 \).

- If \( t \geq T^* \), then \( t = t - 1 \), goto Step 3.
In order to alleviate this computational complexity, R2 is changed to the following heuristic input rule.

HR2: If \( M(t) \neq \emptyset \), let \( x(t) \) be any job matching the machine with the maximum value of \((N_i(t) - 1)T_i\), in \( M(t) \), where \( N_i(t) \) is the number of \( J_i \)-type jobs that have not yet been inputed at time \( t \).

HR2 is a direct extension of the second part of RULE 4 in Algorithm 2 to an \( m \geq 2 \) machine scheduling problem. The rationale of HR2 can be explained as follows. At time \( t \), the finish time of \( x \) can be computed as:

\[
T(x) = \max \left\{ \max_{i \in M(t)} \{ (N_i(t) - 1)T_i + n_i(x, t) \} + t + 1, \right. \\
\left. \max_{i \in M(t)} \{ (N_i - 1)T_i + n_i(x, T(x)) \} \right\}
\]

where \( n_i(x, t) \) is the total number of times \( M_i \) has been missed since \( t \). Clearly, what is inputed at time \( t \) has no effect on the second argument of the max function. For an \( i \in M(t) \), if \( x(t) = i_0 \), \( n_i(x, t) \) will remain unchanged at time \( t \), and all other \( n_i(x, t) \) are increased by 1. Thus, to minimize \( T(x) \), a job matching the machine with the maximum value of \( N_i((t) - 1)T_i \) should be inputed first. However, this is only a one-step optimization, and thus leads to an overall suboptimal (rather than optimal) solution.

Since more than one machine may have the same largest value of \( (N_i(t) - 1)T_i \) at time \( t \), R1 and HR2 might not determine a unique input sequence. To counter this problem, let \( \Omega_2 \) denote the set of all feasible input sequences satisfying R1 and HR2. In case \( |\Omega_2| > 1 \), there are three ways to choose a suboptimal sequence.

Method 1. Choose an arbitrary \( x \in \Omega_2 \) to be the solution.

Method 2. Similarly to Algorithm 3, search \( \Omega_2 \) for an input sequence with the minimum finish time.

Method 3. Partially search \( \Omega_2 \) as far as desired, e.g., search for a certain period of time or a certain number of sequences and choose the best sequence among those searched to be the solution.

The obvious advantage of Method 1 is its simplicity, though the solution obtained might not be as good as that of Method 2, which requires more computational efforts. Method 3 is a compromise between these two.

In order to evaluate the goodness of the heuristic rule described above, we simulated and compared the solutions with the following lower bound of the finish time

\[
T_b = \max \left\{ T_{j1} \right\}_{i, j=1, \ldots, m}
\]

where \( T_{j1} \) is the lower bound of the finish time determined by Algorithm 1 considering \( M_i \) and \( M_j \) only, and \( T_b = \sum_{i=1}^{m} N_i \) is the total number of jobs. Since only one job can be inputed to an assembly line during each unit of time, \( T_h \) is a lower bound of the finish time. Thus, the maximum of two lower bounds, \( T_b \), is also a lower bound. System parameters of the assembly line to be simulated are as follows. Number of machines \( m = 7 \). Length of the conveyor \( T_2 = 16 \). Machines are installed along the conveyor at positions 2, 4, 6, 8, 10, 12, 14 respectively. It takes one unit time to transfer an unfinished job from job exit to job entrance. Simulation results are summarized below.

1. Randomly choose machine times ranging from 1 to 20 and job numbers ranging from 1 to 50. Using Method 1, the simulation results turned out to be surprisingly good. Of 1000 randomly generated examples, only in one example the finish time obtained by Method 1 was 0.05% larger than the lower bound \( T_b \). The lower bound was reached in all other examples. Thus, one can conclude: (1) in most cases, the input sequences determined by Method 1 are optimal, and (2) in most cases, the lower bound \( T_b \) is a good estimate of the optimal finish time.

2. Method 1 was found to perform worst when \((N_i - 1)T_i \) has an identical value for all \( i \). In order to study these worst cases, we generated examples by randomly choosing machine times \( T_i \) between 1 and 20, and letting \( N_i \) be the integer nearest to \( 300/T_i \) on average. The finish times obtained by Method 1 were 23.05% larger than the lower bound. In the worst case, the finish time was 23.05% larger than the lower bound. If an arbitrary \( x_i \in \Omega_1 \) was chosen to be the solution, i.e., HR2 was not used (this is a kind of simple-minded heuristic), the finish times turned out to be 23.05% larger than the lower bound. Thus, heuristic rule HR2 can improve the solution by almost 10 times on the average.

3. Searching \( \Omega_2 \) is much simpler than searching \( \Omega_1 \). For instance, it needs to search only 1128 sequences in \( \Omega_2 \) and use 82.7 seconds of CPU time to find the same optimal solution for the three machine—three job type problem discussed earlier, which needed to search 74308 sequences in \( \Omega_1 \) and use 1851.7 seconds of CPU time. The drawback of Method 2 lies in that the solution obtained may not be overall optimal and it is still time-consuming for large-size
problems. The latter can be alleviated by using Method 3 which can improve the results obtained from Method 1. Consider an example with 41 = 17, 42 = 17, 43 = 20, 44 = 40, 45 = 40, 46 = 29, 47 = 12, 48 = 12, 49 = 5, 50 = 10, 51 = 5, 52 = 5, 53 = 7, the lower bound 54 = 203, and the finish time obtained from Method 1 is 230, which is 13.30% larger than 55. Using Method 3, the finish time is reduced to 227 (11.82% larger than 56) after searching 57953 sequences.

4.2 On Scheduling \( m > 2 \) Machines Processing Multi-Operation Jobs

This is the most difficult problem, where the difficulty mainly arises from the existence of multi-operation type jobs. For example, the best way of processing a single \( J_{123} \)-type job is to process it in one cycle. However, processing all multi-operation jobs in a minimal number of cycles may not result in an overall optimal input sequence. This fact can be seen from the fact that while a multi-operation job is being processed by one machine, the machine that will execute the job’s next operation may have to wait (thus idle) so as to ensure itself to be free when the job arrives at its gate. For the two machine scheduling problem, \( n_{1} \) was found to be the optimal number of \( J_{12} \)-type jobs that should be processed in one cycle. By contrast, it appears to be practically impossible to derive such a number for the case of \( m > 2 \) machines processing multi-operation jobs. Thus, in what follows, we shall derive two domination rules based on which a heuristic algorithm for this general problem will be proposed.

Theorem 4. Let \( \Omega_{3} \) be the set of all feasible input sequences satisfying following input rules:

\[
\begin{align*}
\text{MR1}: & \quad \text{if } \mathcal{M}(t) = \emptyset, \ z(t) = 0. \\
\text{MR2}: & \quad \text{a job of type } J_{i_{1}\ldots i_{k}} \text{ should be inputed before a job of type } J_{i_{1}\ldots i_{l}}, \ where \ i_{1} \geq 1, i_{k+1} < l. \\
\text{Then: } & \quad \Omega_{3} = \emptyset.
\end{align*}
\]

The following heuristic rule can be used to select an input sequence \( z \) from \( \Omega_{3} \).

\[
\begin{align*}
\text{MH1: } & \quad \text{at time } t, \ \text{input a job such that the cost function } C(z,t) = \sum_{i \in \mathcal{M}(t)} (N_{i}(t) - 1)T_{i}\tilde{n}_{i}(z) \\
& \quad \text{is minimized, where } \mathcal{M}(t) \text{ is the set of machines that } P_{0}(t) \text{ matches at time } t, \ N_{i}(t) \text{ is the total number of jobs weakly matching } M_{i} \text{ which have not yet been inputted by time } t \text{ plus jobs weakly matching } M_{i} \text{ that have been inputted before time } t \text{ but will leave the conveyor without } M_{i} \text{ 's processing, } \\
& \quad \tilde{n}_{i}(z) = 1 \text{ if } M_{i} \text{ is missed at } t, \text{ and } \tilde{n}_{i}(z) = 0 \text{ otherwise.}
\end{align*}
\]

MH1 can be explained as follows.

1. In order to reduce the finish time \( T(z) \), at time \( t \) (1) the machine with the largest value of \((N_{i}(t) - 1)T_{i}\) should not be missed (see the explanation of HR2), and (2) the number of machines missed at time \( t \) should be as small as possible. However, these two requirements may not be met simultaneously. So, we construct a cost function \( C(z,t) \) whose

value increases if either a machine with the largest value of \((N_{i}(t) - 1)T_{i}\) is missed or the number of machines missed increases. Thus, minimization of \( C(z,t) \) is equivalent to making a compromise in meeting both requirements (1) and (2).

2. Since the number of job types is finite (and usually not large), \( C(z,t) \) can be minimized by considering job types one by one. If MH1, together with MR1 and MR2, still cannot determine a unique job type, one can simply input a job of one of these types which requires the maximal number of operations. This can reduce the possibility of short supply of jobs.

3. If all jobs are of single-operation type, then for all \( i \in \mathcal{M}(t) \), at most one \( \tilde{n}_{i}(z) = 0 \), and all others have the value of 1. Thus, minimization of \( C(z,t) \) requires a job matching the machine with the largest value of \((N_{i}(t) - 1)T_{i}\) in \( \mathcal{M}(t) \) to be inputed at \( t \). This is the same as HR2.

5 Conclusion

In this paper, we have formulated and solved the problem of scheduling operations of an automatic assembly line. Optimal and suboptimal solutions — that require very little computation — are derived for assembly lines with 2 machines as well as \( m > 2 \) machines processing single-operation jobs by minimizing the total finish time. A heuristic approach is also proposed for the case of \( m > 2 \) machines processing multi-operation jobs. Evaluation of this heuristic approach is our next step of research, which will be reported in a forthcoming paper.

References


