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 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 polynomial 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. Sriskandarajah *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'Gorman *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

[1]The work reported in this paper was supported in part by the National Science Foundation under Grant No. DMC–878721492. Any opinions, findings, and recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the NSF.

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, $M_1, M_2, ..., M_m$, a job entrance and a job exit. There are T_c 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.

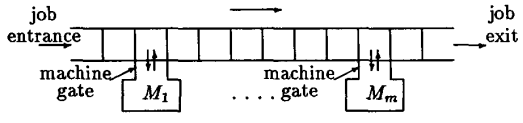

Fig. 1. Configuration of an automatic assembly line.

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, $1 \leq i_1, i_2, ..., i_k \leq m$, $k \geq 1$. For example, each job of type J_{132} 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_f 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. A job is said to *weakly match* a machine if one of its remaining operations 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 $J_{i_1 i_2 \ldots i_k}$–type job will become a $J_{i_2 \ldots i_k}$–type job after it is processed by M_{i_1}.

Definition 3: Let $P_0(t)$ 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 machine is said to be *missed* at time t if (i) $P_0(t)$ matches the machine, and (ii) no job is inputed to the machine at the time $P_0(t)$ reaches the machine. A *collision* is said to occur at time t if $P_0(t)$ 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 $P_0(t)$ 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 $x = \{x(1), x(2), ...\}$, where $x(t)$ represents the type of job inputed at time t. $x(t) = 0$ if no job is inputed at time t. Let $T(x)$ denote the total finish time for an input sequence x and Ω 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 Ω. F is said to *dominate* G if $\forall x \in G$, $\exists x_1 \in F$ such that $T(x_1) \leq T(x)$. The set of optimal input sequences, denoted by S^*, is defined as: $S^* = \{x : x \in \Omega, T(x) \leq T(x_1), \forall x_1 \in \Omega\}$.

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_{12}. 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_{21}–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_{121}–type job: it is a composition of a J_{12}–type job and a J_1–type 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_{12}) 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_{12}) problem is derived to construct and prove an optimal scheduling algorithm in the next subsection.

Let N_1, N_2, N_{12} be respectively the number of jobs of type J_1, J_2, J_{12}, and suppose $N_1 + N_{12} > 0$, $N_2 + N_{12} > 0$. 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 $n_i(x,t)$ denote the total number of times M_i is missed during $[1, t-1]$. Then, the last job matching M_i will be inputed to the assembly line at the time

$$T_{m_i}(x) \equiv (N_i + N_{12} - 1)T_i + n_i(x, T(x)) + 1 \quad i = 1, 2$$

Let $T_m(x) = \min\{T_{m_1}(x), T_{m_2}(x)\}$. Then, from Definition 3 and Definition 5, t is a collision time iff (1) $P_0(t)$ weakly matches both machines, and (2) $t \le T_m(x)$.

Let $\tau_1 = k_1T_1 + 1 = \ell_1T_2$, $\tau_2 = k_2T_1 = \ell_2T_2 + 1$ and $\tau_{12} = k_{12}T_1 = \ell_{12}T_2 > 0$, where $k_1, k_2, k_{12}, \ell_1, \ell_2, \ell_{12}$ are the smallest nonnegative integers (or ∞ if not exist) satisfying these equations. The physical meaning of τ_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 $[t_0+1, t_0+\tau_1]$: $E_1 = \{M_1$ is missed once$\}$, $E_2 = \{M_2$ is missed once$\}$ and $E_{12} = \{$a collision happens without machine miss$\}$. This fact can be verified by observing that if neither E_1 nor E_2 happens in $[t_0+1, t_0+\tau_1-1]$, then $t_0 + \tau_1$ is a collision time. Thus, at least one of E_1, E_2, E_3 will happen at $t_0 + \tau_1$. Similarly, the physical meaning of τ_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 and E_3 will happen during $[t_0 + 1, t_0 + \tau_2]$. If t_0 is a collision time without machine miss, then at least one of E_1, E_2 and E_3 will happen duing $[t_0 + 1, t_0 + \tau_{12}]$. Roughly speaking, $\tau_1(\tau_2, \tau_{12})$ can be thought as the maximum interval between the occurence of $E_1(E_2, E_{12})$ and that of one of E_1, E_2 and E_{12}. Thus, none of E_1, E_2 or E_3 could happen after time $n_1(x,T(x))\tau_1 + n_2(x,T(x))\tau_2 + n_{12}(x,T(x))\tau_{12}$, where $n_{12}(x,t)$ 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 $x \in \Omega$, $n_1(x,T(x))$, $n_2(x,T(x))$ and $n_{12}(x,T(x))$ must satisfy the following inequalities:

(1) $n_1(x,T(x))\tau_1 + n_2(x,T(x))\tau_2 + n_{12}(x,T(x))\tau_{12} \ge T_m(x)$.

(2) $n_1(x,T(x))\tau_1 + n_2(x,T(x))\tau_2 \ge T_1$.

(3) $n_{12}(x,T(x)) \le N_{12}$.

For the limitation of space, all proofs are not presented in this paper. Theorem 1 specifies some constraints on the values of $n_1(x,T(x)), n_2(x,T(x)), n_{12}(x,T(x))$. Given $n_1(x,T(x)), n_2(x,T(x)), n_{12}(x,T(x))$, the finish time of x can be computed as

$$\begin{aligned} T(x) &= \max\{(N_1 + N_{12} - 1)T_1 + n_1(x,T(x)), \\ &\quad (N_2 + N_{12} - 1)T_2 + n_2(x,T(x))\} + 1. \end{aligned}$$

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

$$\min_{n_1,n_2}\{\max\{(N_1 + N_{12} - 1)T_1 + n_1, (N_2 + N_{12} - 1)T_2 + n_2\} + 1.$$

subject to

$$\begin{aligned} n_1\tau_1 + n_2\tau_2 + n_{12}\tau_{12} \;\ge\; &\min\{(N_1 + N_{12} - 1)T_1 + n_1, \\ &(N_2 + N_{12} - 1)T_2 + n_2\} + 1\}. \end{aligned}$$

$$n_1\tau_1 + n_2\tau_2 \ge T_1, \quad n_{12} \le N_{12}.$$

This problem can be easily solved by the following algorithm.

Algorithm 1

Step 1. Set $n_1 = n_2 = n_{12} = 0$. Compute τ_1, τ_2 abd τ_{12} (replace with a large positive integer if not exist).

Step 2. If $n_1\tau_1 + n_2\tau_2 + n_{12}\tau_{12} \ge \min\{(N_1 + N_{12} - 1)T_1 + n_1, (N_2 + N_{12} - 1)T_2 + n_2\} + 1$, then stop. The solution is $\max\{(N_1 + N_{12} - 1)T_1 + n_1, (N_2 + N_{12} - 1)T_2 + n_2\} + 1$ and n_1, n_2, n_{12} are the minimizers. Otherwise, goto Step 3.

Step 3. If $n_1\tau_1 + n_2\tau_2 \ge T_1$ and $n_{12} < N_{12}$, then let $n_{12} = n_{12} + 1$, goto Step 2. Otherwise, goto Step 4.

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

Since $\tau_1 \ge 1$, this algorithm will terminate in a finite number of steps. Let T^* denote the solution and n_1^*, n_2^*, n_{12}^* 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_{12}) problem.

Algorithm 2

Step 1. Apply Algorithm 1 to determine the minimizer n_{12}^*.

Step 2. Generate an input sequence x using the following input rules at time t.

RULE 1: If $P_0(t)$ does not match any machine, $x(t) = 0$.

RULE 2: If $P_0(t)$ matches only M_2, $x(t) = J_2$.

RULE 3: If $P_0(t)$ matches only M_1, then $x(t) = J_{12}$ if $N_{12}(t) > n_{12}^*$, and $x(t) = J_1$ otherwise.

RULE 4. If P_0 matches both machines, then

- If $t > T_1$ and $N_{12}(t) > 0$, then $x(t - T_1) = J_{12}$, $n_{12}^* = n_{12}^* - 1$, and $x(t) = J_{12}$ if $N_{12}(t) > n_{12}^*$, and $x(t) = J_1$ otherwise.
- If $t \leq T_1$ or $N_{12}(t) = 0$, then
  - If $(N_{12}(t) + N_1(t) - 1)T_1 < (N_{12}(t) + N_2(t) - 1)T_2$, then $x(t) = J_2$.
  - If $(N_{12}(t) + N_1(t) - 1)T_1 \geq (N_{12}(t) + N_2(t) - 1)T_2$, then $x(t) = J_{12}$ if $N_{12}(t) > n_{12}^*$ and $x(t) = J_1$ otherwise.

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 inputed at time t plus those that are inputed 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. (We will later remark on this assumption.)

Theorem 2. The input sequence x 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 inputed twice. Recall that a J_{12}–type job becomes a J_2–type job after being processed by M_1. If all the J_2–type jobs at the job entrance have been inputed 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 inputed 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 inputed as early as possible, thus reducing the possibility of short supply. If this still cannot prevent short supply of jobs, we can simply let $x(t) = 0$ 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 on-line algorithm. No pre-computation is necessary and the algorithm can be readily adapted to changes in the number of jobs to be processed. For instance, 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_{12} be the total numbers of J_1, J_2, J_{12} 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 inputed first. For instance, a J_{21}–type job needs two cycles and a J_{2121}–type job needs three cycles to complete. If Algorithm 2 happens to yield $x(t) = J_2$, then a J_{2121}–type job should be inputed. Recall that inputing 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 Ω 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, $\mathcal{M}(t)$, of machines which $P_0(t)$ matches at time t.

R1: If $\mathcal{M}(t) = \emptyset$, then $x(t) = 0$.

R2: If $\mathcal{M}(t) \neq \emptyset$, then let $x(t)$ be any job type that matches a machine in $\mathcal{M}(t)$.

Clearly, in case $|\mathcal{M}(t)| \geq 2$, R2 cannot determine a unique value for $x(t)$. Thus, there could be more than one feasible input sequence satisfying R1 and R2. Let Ω_1 be the set of all such sequences. Then, we have the following theorem.

Theorem 3. Ω_1 dominates Ω.

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 x^* in Ω_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.

- If all the jobs have been inputed, then $T^* = t-1$, $x^* = \{x(t), t = 1, 2, ..., T^*\}$, $t = t-1$. Goto Step 3.
- Otherwise, let $\mathcal{J}(t)$ be the set of all job types matching $\mathcal{M}(t)$. Goto Step 4.

Step 3. If $t = 0$, stop. The optimal sequence is x^*. Otherwise, goto Step 4.

Step 4. If $\mathcal{J}(t) = \emptyset$, then $x(t) = 0$, goto Step 2. Otherwise, let $x(t) = J \in \mathcal{J}(t)$, delete J from $\mathcal{J}(t)$, and goto Step 2.

A program has been written in LISP to implement Algorithm 3. Theoretically, this program can be used to find optimal solutions to all single-operation job type scheduling problems. However, due to its computational complexity, this program is useful only for problems with very small numbers of machines and jobs. For instance, a three machine–three job type problem with job numbers $N_1 = 17, N_2 = 25, N_3 = 13$ and machine times $T_1 = 6, T_2 = 4, T_3 = 8$ needs to search a total of 3540 input sequences and use 182.7 seconds of CPU time on an Apollo DN3000 workstation to find an optimal solution. If the numbers of jobs are increased to $N_1 = 25, N_2 = 37, N_3 = 17$, a total of 74308 input sequences need to be searched using 1851.7 seconds of CPU time.

In order to alleviate this computational complexity, R2 is changed to the following heuristic input rule.

HR2: If $\mathcal{M}(t) \neq \emptyset$, let $x(t)$ be any job matching the machine with the maximum value of $(N_i(t) - 1)T_i$ in $\mathcal{M}(t)$, 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 $m > 2$ machine scheduling problem. The rationale of HR2 can be explained as follows. At time t, the finish time of x can be computed as:

$$T(x) = \max\{ \begin{array}{ll} \max_{i \in \mathcal{M}(t)} & \{(N_i(t) - 1)T_i + n_i'(x,t)\} + t + 1, \\ \max_{i \notin \mathcal{M}(t)} & \{(N_1 - 1)T_i + n_i(x, T(x))\}\}. \end{array}$$

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 $i_0 \in \mathcal{M}(t)$, if $x(t) = i_0$, $n_{i_0}'(x,t)$ will remain unchanged at time t, and all other $n_i'(x,t), i \in \mathcal{M}(t)$ 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 Ω_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 Ω_2 for an input sequence with the minimum finish time.

Method 3. Partially search Ω_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

$$T_b = \max\{T_{b1}, \max_{i \neq j,\ i,j=1,...,m} T_{ij}^*\}$$

where T_{ij}^* is the lower bound of the finish time determined by Algorithm 1 considering M_i and M_j only, and $T_{b1} = \sum_{i=1}^m N_i$ is the total number of jobs. Since only one job can be inputed to an assembly line during each unit of time, T_{b1} 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 $T_c = 16$. Machines are installed along the conveyor at positions $2, 4, 6, 8, 10, 12, 14$ respectively. It takes one unit time to trnasfer 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 0.6% 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 $200/T_i + 1$. Thus, $(N_i - 1)T_i + 1$ approximately equals 200. 1600 such examples were studied. On average, the finish times obtained by Method 1 were 2.86% larger than the lower bound. In the worst case, the finish time was 13.30% larger than the lower bound. If an arbitrary $x_1 \in \Omega_1$ 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 Ω_2 is much simpler than searching Ω_1. For instance, it needs to search only 1128 sequences in Ω_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 Ω_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 too 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 $N_1 = 17, N_2 = 17, N_3 = 40, N_4 = 20, N_5 = 40, N_6 = 40, N_7 = 29, T_1 = 12, T_2 = 12, T_3 = 5, T_4 = 10, T_5 = 5, T_6 = 5, T_7 = 7$, the lower bound $T_b = 203$, and the finish time obtained from Method 1 is 230, which is 13.30% larger than T_b. Using Method 3, the finish time is reduced to 227 (11.82% larger than T_b) after searching 53953 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_{12}^* 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 Ω_3 be the set of all feasible input sequences satisfying following input rules:

MR1: If $\mathcal{M}(t) = \emptyset$, $x(t) = 0$.

MR2: A job of type $J_{i_1 \ldots i_k i_{k+1} \ldots i_{k+l}}$ should be inputed before a job of type $J_{i_1 \ldots i_k}$ where $l \geq 1$, $i_{k+1} < i_k$.

Then, $\Omega_3 \Rightarrow \Omega$.

The following heuristic rule can be used to select an input sequence x from Ω_3.

MH1: At time t, input a job such that the cost function

$$C(x,t) = \sum_{i \in \mathcal{M}(t)} (N_i(t) - 1) T_i n_i^t(x)$$

is minimized, where $\mathcal{M}(t)$ is the set of machines that $P_0(t)$ matches at time t, $N_i(t)$ is the total number of jobs weakly matching M_i which have not yet been inputed by time t plus jobs weakly matching M_i that have been inputed before time t but will leave the conveyor without M_i's processing, $n_i^t(x) = 1$ if M_i is missed at t, and $n_i^t(x) = 0$ otherwise.

MH1 can be explained as follows.

1. In order to reduce the finish time $T(x)$, 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 $C(x,t)$ 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 $C(x,t)$ 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), $C(x,t)$ 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 $n_i^t(x) = 0$, and all others have the value of 1. Thus, minimization of $C(x,t)$ 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 sloved 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

[1] K. R. Baker, *Introduction to Sequencing and Scheduling*, Wiley & Sons, 1974.

[2] L. D. Burns and C. F. Daganzo, "Assembly line job sequencing principles," *The International Journal of Production Research*, vol. 25, no. 1, pp. 71–99, 1987.

[3] S. French, *Sequencing and Scheduling*, Halsted Press, 1972.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability–A Guide to the Theory of NP–Completeness*, W. H. Freeman and Company, New York, 1979.

[5] S. Ghosh and R. J. Gagnon, "A comprehensive literature review and analysis of the design, balancing, and scheduling of assembly systems," *The International Journal of Production Research*, vol. 27, no. 4, pp. 637–670, 1989.

[6] T. E. Morton and T. L. Smunt, *A Planning and Scheculing System for Flexible Manufacturing*, pp. 151–159, Elsevier Science Publishers B. V. (North Holland), 1986.

[7] P. O'Gorman, J. Gibbons, and J. Browne, *Evaluation of Scheculing Systems for a Flexible Transfer Line using a Simulation Model*, pp. 209–221, Elsevier Science Publishers B. V. (North Holland), 1986.

[8] C. Sriskandrajah, P. Ladet, and R. Germain, *Scheduling Methods for a Manufacturing System*, pp. 173–189, Elsevier Science Publishers B. V. (North Holland), 1986.