A Cutting and Scheduling Problem in Float Glass Manufacturing

Byungsoo Na[1], Shabbir Ahmed, George Nemhauser and Joel Sokol
H. Milton Stewart School of Industrial and Systems Engineering,
Georgia Institute of Technology,
765 Ferst Drive, Atlanta, GA 30332
{byungsoo.na, sahmed, gnemhaus, jsokol}@isye.gatech.edu

Abstract

This paper considers a cutting and scheduling problem of minimizing scrap motivated by float glass manufacturing and introduces the *FGSP* (float glass scheduling problem). We relate it to classical problems in the scheduling literature such as no-wait hybrid flow shops and cyclic scheduling. We show that the problem is NP-hard, and identify when each of the problem's components are polynomially solvable and when they induce hardness. In addition, we propose a simple heuristic algorithm, provide its worst-case performance bounds, and demonstrate that the bounds are tight. When the number of machines is two, the worst-case performance is $\frac{5}{3}$.

Key words: Cutting, Scheduling, Float line, Glass, Hybrid flow shop, Cyclic schedule

1 Introduction

Flat glass manufacturing is a continuous process whereby a ribbon of molten glass is produced in a furnace and then cooled on a bath of molten tin to ensure flatness. The continuous glass ribbon is then carried on rollers through an annealing lehr, machine-cut according to customer size requirements, and offloaded for distribution by automated machines.

The processing time to cut a rectangular unit of glass (called a *plate*) is proportional to its size. Moreover, it takes a constant amount of time (independent of size) for an offloading machine to pick up, move and release the glass to a container, and return. Thus, some glass will be wasted if it is cut but cannot be picked up before it gets to the end of the conveyor because the offloading machines are all busy. This wasted glass is called *scrap*. Given a set of jobs, each consisting of a size of glass and a number of units of that size, the objective is to sequence the production of the jobs to minimize scrap.

Because of the limitations of the glass-making process and the equipment involved, there are several operational restrictions.

1. Continuous time production This implies that even if no offloading machine is available to pick glass, the glass will still be produced and therefore wasted as scrap.

[1]Corresponding author: Byungsoo Na (Phone) 1-404-894-2320 (Fax)1-404-894-2301, byungsoo.na@isye.gatech.edu

2. Identical machines with constant service time Each offloading machine has the same cycle time (the time it takes to pick a plate, put it into the container, and return to the ready position).

3. Multi-unit products Jobs vary in both plate size and the number of units required.

4. Machine dedication Each offloading machine can deal with only one container of glass at a time. Since each container stores glass of only one customer order (one job), all units of a job must be assigned to the same offloading machine.

5. No preemption Once an offloading machine begins to process a job, it must complete all of the units of that job before it can begin to process another job.

The problem of minimizing the amount of wasted glass in float glass manufacturing is a complicated optimization problem in which cutting and sequencing must be considered simultaneously. In Na et al. (2012), we developed a heuristic algorithm for solving such problems. Using real data from a float glass manufacturer, we demonstrated empirically that the algorithm produced very high quality solutions quite rapidly and therefore provided the needed tool for daily scheduling.

The real manufacturer's policies included several additional operational restrictions that add extra complexity on top of the core problem. In this paper we theoretically study the core float glass scheduling problem. We relate it to classical problems in the literature, study its computational complexity, and provide performance bounds on the quality of solutions obtained by a polynomial time heuristic.

2 Float Glass Scheduling Problem

2.1 Problem Statement

In float glass manufacturing, there is a set of jobs J to be processed in two stages, stage 1 (the cutting process) and then stage 2 (the offloading process). Stage 1 has one machine and stage 2 has m identical parallel machines. Job j has n_j, units that must be produced. The processing time per unit of job j in stage 1 is t_j (the cutting time of job j) and the processing time of a unit of every job in stage 2 is T (by the constant machine cycle time property.) No intermediate storage exists between stages 1 and 2. The property of continuous production characterizes no-wait scheduling. The objective is to minimize scrap. We call this problem the **FGSP** (*float glass scheduling problem*).

Next we show that FGSP is equivalent to the problem of minimizing the completion time of the cutting machine.

Proposition 1. *Minimizing scrap is equivalent to minimizing the completion time of the cutting machine.*

Proof. Since glass is continuously passing through the cutting machine at a constant rate, the completion time of the cutting machine is the sum of the time for cutting ordered glass and the time for cutting scrap. The result then follows immediately since the time for cutting ordered glass is a constant and the time for cutting scrap is proportional to the amount of scrap. □

Therefore, in scheduling terminology, FGSP is a no-wait hybrid flowshop problem (see Gupta and Tunc (1991) and Linn and Zhang (1999)) with parallel machines at stage 2, whose objective is to minimize completion time subject to some additional restrictions. Its scheduling notation is $F_2 \mid \text{no wait}, m_1 = 1, m_2 = m \geq 2 \mid C_{\max}$ where m is the number of parallel machines at stage 2. The restrictions are constant processing time in stage 2 and *machine dedication* and *no preemption* with respect to the multi-units of a given job (otherwise each unit could be considered as an individual job).

Worst case performance analysis has been studied for no-wait (or blocking) flowshops with parallel machines. Sriskandarajah (1993) proved that a list scheduling algorithm where each job is processed in the order in which it appears on the list has a worst case bound of $3-\frac{1}{m}$ and if jobs are scheduled in stage 2 in non-increasing order of processing time, the worst case bound decreases to 2. The *multi-unit products* and *machine dedication* constraints make FGSP harder. However, it is not comparable to $F_2 \mid \text{no wait}, m_1 = 1, m_2 = m \geq 2 \mid C_{\max}$ because of the restriction of constant processing time in stage 2 of FGSP.

Other related literature concerns *cyclic scheduling*. When a set of jobs is produced in a no-wait flowshop and each job has multiple units, the same schedule is repeated over and over again. This repeated pattern is called a *cyclic schedule* in operations research (see McCormick et al. (1989) or Pinedo (2008)) and a *campaign* in chemical processes (see Birewar and Grossmann (1989-I) and Birewar and Grossmann (1989-II)). FGSP also yields cyclic schedules as will be explained below because the *machine dedication* and *no preemption* restrictions enforce that the same schedule should be repeated. However, the type of cycles that appear in FGSP has a different structure than those considered previously.

A cycle in FGSP consists of a set of no more than m jobs processed simultaneously in rotation. We call such a set a *covey*. For example, for $m = 3$, given a covey of jobs $\{i, j, k\}$, in stage 1 a unit of i is cut, followed by a unit of j and then a unit of k and then again a unit of i, etc. All of the i units are offloaded by the first machine in stage 2, all of the j units by the second machine, etc. The covey ends when all of the units of one of the jobs, say i, are completed. At this time, a new job, say h, can be started yielding the new covey $\{h, j, k\}$. This continues until all jobs are done. Observe that if the sum of the cutting times t_j of the jobs in a covey exceeds the processing time T in stage 2, there will be no waste. However, if the sum of the cutting times in a covey is less than T, there will be waste that is proportional to $\left(T - \sum_{j \in \text{covey}} t_j\right)$ and the number of rotations in the covey. The following example is provided to aid the understanding of the definition of a *covey*.

Example: Table 1 introduces an example set of four jobs. We assume that two offloading machines

Table 1: Four jobs are to be produced.

index	cutting time (seconds)	number of units
A	6	100
B	4.5	150
C	3	100
D	8	80

are available and that their cycle time is 10 seconds. In this example, we define three coveys. The first covey (Covey P) consists of jobs A and B. Job A is offloaded by machine 1 and job B is offloaded by machine 2. After 100 units of this covey, job A is completed and is replaced by job C. Thus, jobs C and B define the second covey (Covey Q). After 50 units of Covey Q, job B is completed (a total of 150 units) and is replaced by job D. Jobs C and D thus define the third covey (Covey R). The Gantt chart in Figure 1 illustrates this schedule for each machine. The top row in the figure shows the schedule for the cutter, and the other two rows show the schedule for each offloading machine. Note that in Covey Q, the sum of processing times is 7.5 seconds, which

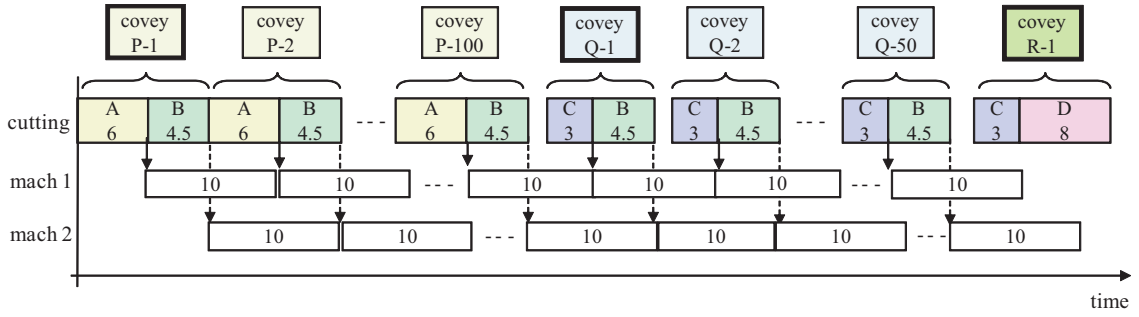


Figure 1: Gantt chart of coveys.

is 2.5 seconds less than the cycle time. Thus, the cutting process has 2.5 seconds of blocking in each rotation of Covey Q, and the result is an amount of scrap proportional to (2.5 seconds) $\times$ (the number of rotations in the covey).

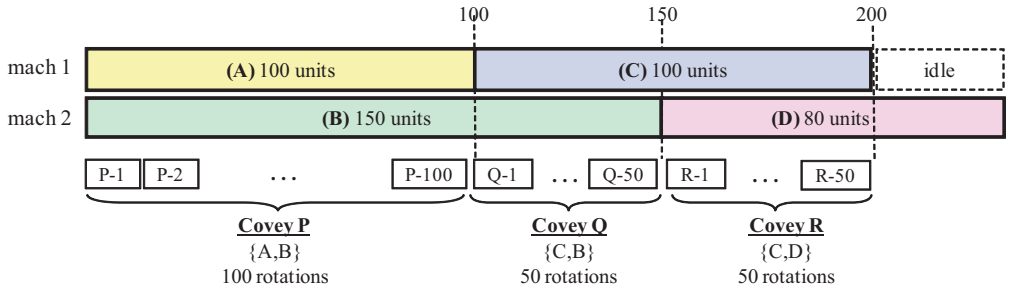


Figure 2: Unit-based view of coveys.

Another useful representation of coveys is a unit-based view illustrated in Figure 2. This view

specifies the sequence of jobs, their assignment to offloading machines, units to be produced of each job and the number of rotations of each covey. Figure 2 shows that machine 1 first offloads 100 units of job A and then offloads 100 units of job C then remains idle. Machine 2 offloads 150 units of job B and 80 units of job D. In this view, it is easy to see the change of coveys; Covey P runs for 100 rotations, followed by 50 rotations of Covey Q and 50 rotations of Covey R (and then 30 rotations of a fourth covey consisting only of job D).

In the above example, each job in a covey appears only once. Such a covey is called a *minimal covey*. A covey in which at least one job appears more than once is called a *mixed covey*. By the *machine dedication* and *no preemption* restrictions, the schedule of FGSP must consist of coveys, which can be minimal or mixed. In Appendix, we provide a rule which divides a mixed covey into several minimal coveys and we prove that such a division process does not increase scrap. Therefore, in FGSP we can restrict the solution space to minimal coveys and for the rest of the paper only schedules with minimal coveys are considered.

We denote the kth covey by C_k, where $C_k \subset J$. The number of rotations of the kth covey is $n(C_k)$. The processing time of one rotation of covey k, denoted by $t(C_k)$, is defined as the sum of stage 1 processing times of one unit of each job in covey k: $t(C_k) = \sum_{j \in C_k} t_j$. Coveys also must satisfy the following restrictions:

(C.1) Units of a Job: The number of units of job j equals the number of rotations of the coveys containing job j: $n_j = \sum_{k:j \in C_k} n(C_k)$.

(C.2) Limited Number of Machines: The number of jobs in a covey is at most the number of stage 2 machines: $|C_k| \leq m, \ \forall k$. The ith element in a covey is processed by the ith machine ($i \leq m$). If the ith element in a covey is empty, it means that the ith machine is idle when the covey is processed.

(C.3) No Preemption of a Product: If job j is an element of both C_p and C_q ($p < q$), then j must be in C_r for all r such that $p < r < q$.

(C.4) Machine Dedication: If job j is assigned to the ith element of a covey in which job j appears for the first time, then j should be assigned to the ith element of all the following coveys that contain job j.

(C.5) Completion of All Jobs: The set J of jobs should be covered by the elements in all coveys: $\bigcup C_k = J$.

In terms of coveys, the objective function is given by

$$\min C_{\max} \quad = \quad \min \sum_k \ \Big(n(C_k) \times \max\{t(C_k), \ T\} \Big).$$

A *schedule* in FGSP is defined as a sequence of coveys where between two consecutive coveys, one or more jobs have finished and/or one or more jobs have started. The optimization problem thus consists of determining a sequence of coveys that produce all jobs such that the amount of scrap is minimized. With a slight abuse of notation, the ith covey in schedule S is represented as $C_i \in S$ in this paper. Once a sequence of coveys is determined, we can easily find implied start and completion times of each job as well as the assignment of jobs to the stage 2 machines.

Recall that in our definition, a covey ends after the rotation in which one of its jobs finishes. In the new covey that begins next, the place of the finished job is taken by a new (possibly empty) job and all other in-progress jobs remain, so that the two consecutive coveys differ by just one job. When two or more jobs coincidentally end at the same time and more than one job enters to form the next covey, additional waste can occur in one rotation during transition of the coveys. In Figure 3, assume that the cycle time of offloading machines is 10 seconds and there are three offloading machines. The first covey consists of jobs A, B, and C, and its sum of processing times is 4+1+5 = 10 (seconds), which does not produce any scrap. Suppose that jobs A and B are finished at the same rotation, and they are replaced by jobs D and E, respectively. Then, the second covey consists of jobs D, E, and C, and its sum of processing times is 2+3+5 = 10 (seconds), which again has no scrap. However, during the transition of coveys, two seconds of additional scrap is incurred. This phenomenon rarely happens and the amount of scrap is negligible for real-world applications. Thus, we ignore this small amount of *transient scrap*.

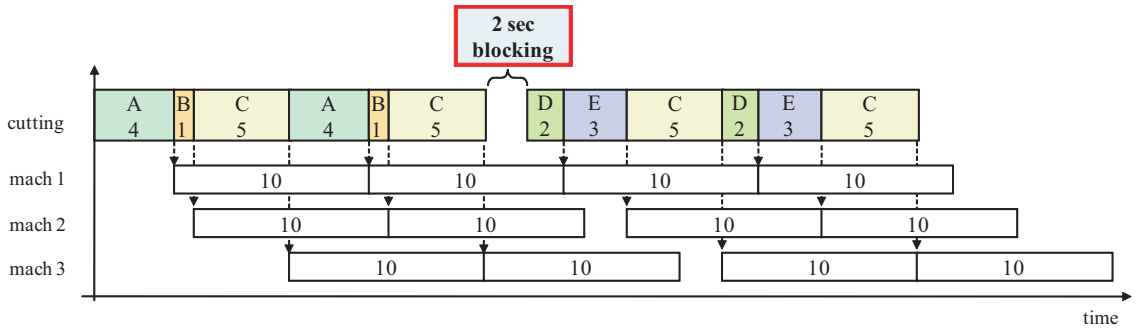


Figure 3: Scrap in coveys transition

3 Complexity

In the previous section, we introduced FGSP and provided its underlying structure, *coveys*. FGSP has two main elements that make it interesting. Jobs require different processing times in stage 1 and jobs may have a different number of units to be produced. If both processing times and number of units are identical for each job, the problem is trivial. In this section, we first consider the complexity of simplified problems obtained by relaxing each of two elements: the *time model* and the *unit model*. They are decision versions of the corresponding optimization problems of

FGSP.

3.1 Time Model

When every job has the same number of units to be produced, but jobs might require different stage 1 processing times per unit, we refer to the model as the *time model*. Using the notation of Garey and Johnson (1979), the time model is stated as follows:

INSTANCE: Finite set J, a time $t_j \in \mathbb{R}^+$ for each $j \in J$, a positive integer m (the number of machines), and positive numbers T (the cycle time) and K.
QUESTION: Can J be partitioned into disjoint subsets $C_1, C_2, \cdots$ such that

$$\begin{aligned} |C_i| &\leq m, \quad \forall i \qquad \text{(at most } m \text{ jobs in each covey)} \\ \sum_i \max\Big\{ \sum_{j \in C_i} t_j,\ T \Big\} &\leq K \qquad\quad \text{(the completion time).} \end{aligned}$$

When the number of machines is two, the time model can be solved in polynomial time using minimum weighted perfect matching. However, we have even a faster algorithm called *match largest and smallest jobs*. For $m = 2$, a simple $O(n \log n)$ algorithm solves the time model where n is the number of jobs.

Algorithm: Match Largest and Smallest Jobs

(Step 1) Sort jobs by non-decreasing order in time per unit. Let $(j_1, j_2, \ldots, j_{n-1}, j_n)$ be the sequence of jobs after sorting.

(Step 2-a) If the number n of jobs is even, make coveys $(j_1, j_n), (j_2, j_{n-1}), \ldots, (j_{\frac{n}{2}}, j_{\frac{n}{2}+1})$.

(Step 2-b) If the number n of jobs is odd, make one covey (j_n) for the job with the largest time, and then make coveys for remaining jobs as in the even case.

Theorem 2. *The algorithm* match largest and smallest jobs *produces an optimal solution in the time model when the number of machines is two.*

The proof of Theorem 2 is given in the Appendix. When the number of machines is three, the time model is NP-complete in the strong sense.

Theorem 3. *The time model is NP-complete in the strong sense when* $m = 3$.

Proof. It is in NP since verifying a solution can be easily checked in polynomial time. We show it is NP-complete by reduction from 3-PARTITION (Garey and Johnson (1979)). In 3-PARTITION, we are given positive integers b and $\bar{m}$ and a set $N = \{1, 2, ..., n\}$ of $n = 3\bar{m}$ elements, each having a positive integer size $a_j (< b)$ such that $\sum_{j=1}^{n} a_j = \bar{m}b$. The problem is to determine whether

there exists a partition of N into $\bar{m}$ subsets, each containing exactly 3 elements from N and such that the sum of the sizes in each subset is b. The solution is *yes* if such a partition exists, and *no* otherwise. In the strong sense, 3-PARTITION is NP-complete.

Consider the instance of the time model in which $J = N$, $t_j = a_j$ $(j \in J)$, $m = 3$, $T = b$, and $K = \bar{m}b$. Then, set J of the time model can be partitioned into disjoint subsets where the sum of elements in each subset is b if and only if 3-PARTITION has a *yes* solution. □

3.2 Unit Model

When every job has the same processing time per unit in stage 1, but jobs might require different numbers of units to be produced, we refer to the model as the *unit model*. Consider the special case where $t_j = t \leq \frac{T}{m}$. Then,

$$\begin{aligned} C_{\max} &= \sum_i n(C_i) \times \max\Big\{ \sum_{j \in C_i} t_j,\ T \Big\} \\ &= \sum_i n(C_i) \times \max\Big\{ |C_i| \times t,\ T \Big\} \\ &= T \times \sum_i n(C_i). \end{aligned}$$

The problem description of this unit model is

INSTANCE: Finite set J, a time $t_j = t \leq \frac{T}{m}$, $\forall j$, a production requirement $n_j \in \mathbb{Z}^+$ for each $j \in J$, a positive integer m (the number of machines), and positive numbers T (the cycle time) and K.
QUESTION: Does J have distinct subsets $C_1, C_2, \cdots$ with the number of rotations $n(C_i) \in \mathbb{Z}^+$ such that

$$T \times \sum_i n(C_i) \quad \leq \quad K$$

Coveys satisfy the restrictions **(C.1)** – **(C.5)** as defined in Section 2.

This unit model is identical to the parallel machine scheduling problem with objective of minimization of makespan. Since one rotation of every covey takes time equal to T, the completion time is determined by the total number of rotations of all coveys. Therefore, the completion time is $T \times \big(\sum_i n(C_i)\big)$, which corresponds to the makespan in the parallel machine scheduling problem.

When the number of machines is at least two, the unit model is NP-complete.

Theorem 4. *The unit model is NP-complete when $m \geq 2$.*

Proof. It is in NP since verifying a solution can be easily checked in polynomial time. We show it is NP-complete by reduction from MULTIPROCESSOR SCHEDULING (Garey and Johnson (1979)). In multiprocessor scheduling, or parallel machine scheduling, we are given set A of tasks,

number $\bar{m} \in \mathbb{Z}^+$ of processors, length $l(a) \in \mathbb{Z}^+$ for each $a \in A$, and a deadline $D \in \mathbb{Z}^+$. The problem is to determine whether there is an $\bar{m}$-processor schedule for A that meets the overall deadline D. It is NP-complete for $\bar{m} \geq 2$.

Consider an instance of the unit model with $J = A, n_j = l(a), m = \bar{m}$ and $K = D$. The completion time is $T \times \left(\sum_i n(C_i)\right)$. The value of $T \times \sum_i n(C_i) \leq K$ if and only if MULTIPROCESSOR SCHEDULING has a *yes* solution. □

Theorem 4, of course, implies that the full model considering both time and unit elements is NP-complete for $m \geq 2$.

4 A Covey-based Scheduling Algorithm

For the size of the FGSP instances that arise in practice, solution methods capable of proving optimality are not practical. In this section, we propose a simple heuristic algorithm, called the *Longest Unit First (LUF)*, and analyze its worst case performance for FGSP. The *Longest Processing Time First (LPT)* algorithm (Graham (1969)) for parallel machine scheduling motivates the proposed *LUF* algorithm.

Algorithm: Longest Unit First (LUF)

(Step 0) Choose the m jobs requiring the largest number of units. The first covey consists of these m jobs. For $i = 1, \ldots, m$, the first-stage machine produces a unit of the ith job in the covey, and it is assigned to the ith machine at the second-stage.

(Step 1) The procedure of producing a unit of each job in the covey is repeated until all of the units of a job in the covey are completed.

(Step 2) The completed job exits from the covey, and a new job with the largest number of units among unassigned jobs enters into the covey, forming a new covey. This entered job is assigned to the second-stage machine to which the exited job was assigned.

(Step 3) Steps 1 and 2 are repeated until all jobs are done.

For example, suppose we have three machines and six jobs: job A requires 170 units, B 120, C 90, D 80, E 50, and F 70 units. First, the three jobs with the most required production, jobs A, B, and C, are assigned to machines. Job A is assigned to machine 1, then job B is assigned to machine 2, and job C is assigned to machine 3. This is the first covey. At this point, all machines are busy for 90 rotations, until job C finishes on machine 3. Job D, having the largest number of units among unassigned jobs, is then assigned to machine 3. Then, job F is assigned to machine 2 after another 30 rotations (120 total), and job E can be assigned to either machine 1 or machine 3 after 50 more rotations (170 total).

With regard to the complexity of the *LUF* algorithm, sorting n jobs takes $O(n \log n)$ time. Since assigning a new job to one of m machines takes $O(m)$ time, assigning all n jobs takes $O(mn)$. Therefore, the overall complexity of the *LUF* algorithm is $O((\log n + m)n)$ where n is the number of jobs and m is the number of machines at the second stage.

Let S be an arbitrary schedule, $\hat{S}$ be a schedule produced by the *Longest Unit First* algorithm, and S^* be an optimal schedule. The sum of rotations for the coveys of schedule S is defined as $N(S) := \sum_{i:C_i \in S} n(C_i)$.

In the next section, we will prove the following theorem for two cases according to $N(S), N(\hat{S})$ and $N(S^*)$.

Theorem 5. *For any schedule S with $N(S) \leq N(S^*)$, we have*

$$C_{\max}(S) \leq \Big(1 + \frac{m-1}{m}\Big) C_{\max}(S^*),$$

where m is the number of second-stage machines. For schedule $\hat{S}$ with $N(\hat{S}) > N(S^*)$, produced by the LUF *algorithm, we have*

$$C_{\max}(\hat{S}) \leq \Big(\ \big(1 + \frac{m-1}{m}\big) \ + \ \big(\frac{1}{3} - \frac{1}{3m}\big) \ \Big) C_{\max}(S^*).$$

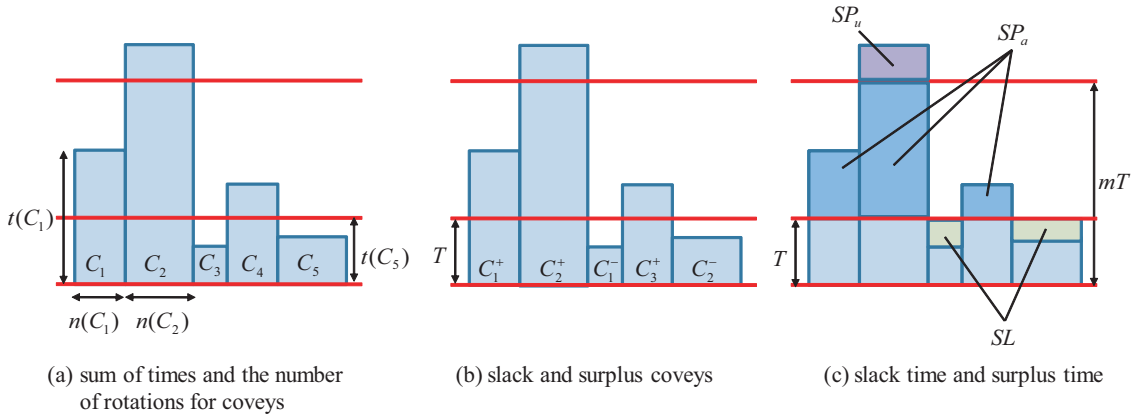
From the first result of Theorem 5, if we minimize the number of rotations of schedule S so that $N(S) \leq N(S^*)$, a worst case bound for FGSP is $\Big(1 + \frac{m-1}{m}\Big)$. This bound is obtained by only considering the unit element of the problem, i.e., minimizing the makespan, which is known to be NP-hard.

4.1 Basic Properties

In this section, we present notation and some basic properties that will be used in the worst case analysis. In Figure 4, the horizontal-axis represents the number of rotations of each covey and the vertical-axis represents the sum of times of jobs in a covey. From Figure 4 (a), we can identify the number of rotations, $n(C_i)$ and the sum of times, $t(C_i)$, as the width and height of the rectangle for each covey C_i.

To calculate the completion time, we analyze the sum of stage 1 machine times $t(C_i)$ and the number of rotations required $n(C_i)$ for each covey C_i of the schedule. We partition coveys $C_1, C_2, \ldots, C_K$ of a schedule S into two sets: a set C^- of slack coveys that incur scrap at each rotation because their total stage 1 processing time is less than T and a set C^+ of surplus coveys whose total stage 1 processing time is at least T per rotation. They are defined as

$$C^- := \{C_i \in S : \sum_{j \in C_i} t_j < T\} \qquad \text{and} \qquad C^+ := \{C_i \in S : \sum_{j \in C_i} t_j \geq T\}.$$



(a) sum of times and the number of rotations for coveys

(b) slack and surplus coveys

(c) slack time and surplus time

Figure 4: Slack coveys and surplus coveys

We denote slack coveys as $C_1^-, C_2^-, \ldots, C_{|C^-|}^-$ and denote surplus coveys as $C_1^+, C_2^+, \ldots, C_{|C^+|}^+$, as illustrated in Figure 4 (b). In addition, we denote the sum of rotations for all coveys in schedule S as $N(S)$, for slack coveys, $N^-(S)$, and for surplus coveys, $N^+(S)$. In other words, $N(S) := \sum_{C_i \in S} n(C_i)$, $N^-(S) := \sum_{C_i^- \in C^-} n(C_i^-)$, and $N^+(S) := \sum_{C_i^+ \in C^+} n(C_i^+)$. Clearly, $N(S) = N^-(S) + N^+(S)$.

Consider the following two lower bounds for the optimal solution.

$$\textbf{Lower Bound 1:} \qquad C_{\max}(S^*) \geq \sum_{j=1}^{J} n_j \; t_j$$

$$\textbf{Lower Bound 2:} \qquad C_{\max}(S^*) \geq N(S^*) \; T + \sum_{j=1}^{J} n_j \; \max(t_j - T, \; 0)$$

The first lower bound is time-based; the completion time is trivially no less than the sum of all stage 1 processing times. The second trivial lower bound, based on units, is that $C_{\max}(S^*) \geq N(S^*)\; T$ since each rotation must take at least T time including slack if necessary. However, we can tighten this bound. If a single job j requires more stage 1 processing time than T, then each rotation of its covey will have at least $t_j - T$ surplus time. Lower Bound 2 includes this amount of surplus time that cannot be eliminated.

In our analysis, we need to calculate the total amount of slack time, the total surplus time up to $(m-1)T$ per rotation and the total surplus time above $(m-1)T$ per rotation in schedule S. For a schedule S, we define $SL(S), SP_a(S)$, and $SP_u(S)$ as the following:

$$
\begin{aligned}
SL(S) &:= \sum_{C_i^- \in C^-} n(C_i^-)\ \Big(T - t(C_i^-)\Big) \\
SP_a(S) &:= \sum_{C_i^+ \in C^+} n(C_i^+)\ \min\Big\{(m-1)T,\ t(C_i^+) - T\Big\} \\
SP_u(S) &:= \sum_{C_i^+ \in C^+} n(C_i^+)\ \max\Big\{0,\ t(C_i^+) - mT\Big\}.
\end{aligned}
$$

The above lower bounds can be represented in terms of slack time and surplus time in the following propositions.

Proposition 6. *For an arbitrary schedule S,*

$$N(S) \cdot T + SP_a(S) + SP_u(S) - SL(S) \leq C_{\max}(S^*). \tag{1}$$

Proof. By definition, the sum of stage 1 processing times for all jobs is equivalent to $N(S) \cdot T$ +(total surplus time) − (total slack time) for any schedule S. Therefore, we have

$$
\begin{aligned}
C_{\max}(S^*) &\geq \sum_{j=1}^{J} n_j\ t_j \qquad\qquad (\because \text{Lower Bound 1}) \\
&= N(S) \cdot T + SP_a(S) + SP_u(S) - SL(S).
\end{aligned}
$$

□

Proposition 7. *For any schedule S with $N(S) \leq N(S^*)$,*

$$N(S) \cdot T + SP_u(S) \leq C_{\max}(S^*).$$

Proof. Because $\sum_{j=1}^{J} n_j\ \max(t_j - T,\ 0) \geq SP_u(S)$, we have

$$
\begin{aligned}
C_{\max}(S^*) &\geq N(S^*) \cdot T + \sum_{j=1}^{J} n_j\ \max(t_j - T,\ 0) \qquad (\because \text{Lower Bound 2}) \\
&\geq N(S) \cdot T + SP_u(S).
\end{aligned}
$$

□

4.2 Worst case bound for any schedule S with $N(S) \leq N(S^*)$

In this section, we formalize a worst case bound on the cost of any schedule that has no more rotations than the optimal schedule.

First, we consider some basic properties of the amount of slack time and surplus time that will be useful in the analysis. By the definition of slack time, we have

$$SL(S) \leq N^{-}(S)\ T \tag{2}$$

because $SL(S) = \sum_{C_i^- \in C^-} n(C_i^-)\big(T - t(C_i^-)\big) \leq \sum_{C_i^- \in C^-} n(C_i^-)T = N^-(S)T$. Similarly, by the definition of surplus time, we have

$$SP_a(S) \leq (m-1)\ N^{+}(S)\ T \tag{3}$$

because $SP_a(S) = \sum_{C_i^+ \in C^+} n(C_i^+) \min\big\{(m-1)T,\ t(C_i^+) - T\big\} \leq \sum_{C_i^+ \in C^+} n(C_i^+)(m-1)T = (m-1)N^+(S)T$.

Proposition 8. *For any schedule S with $N(S) \leq N(S^*)$,*

$$C_{\max}(S) \leq \Big(1 + \frac{m-1}{m}\Big) C_{\max}(S^*).$$

Proof. We consider two cases: $SL(S) > SP_a(S)$ and $SL(S) \leq SP_a(S)$.

Case (i): $SL(S) > SP_a(S)$
By Proposition 7, we have

$$N(S)\ T + SP_u(S) \leq C_{\max}(S^*). \tag{4}$$

Since $SL(S) \leq N^-(S)\ T$ (inequality (2)) and $SL(S) > SP_a(S)$, we have $(m-1)\ SP_a(S) < (m-1)\ SL(S) \leq (m-1)\ N^-(S)\ T$. By inequality (3), $SP_a(S) \leq (m-1)\ N^+(S)\ T$. Therefore, the sum of these two inequalities yields

$$m\ SP_a(S) < (m-1)\ \big(N^-(S) + N^+(S)\big)\ T = (m-1)\ N(S)\ T.$$

Because we know that $N(S)\ T \leq N(S^*)\ T \leq C_{\max}(S^*)$, we have

$$SP_a(S) < \frac{m-1}{m}\ C_{\max}(S^*). \tag{5}$$

By inequalities (4) and (5), we have

$$N(S)\ T + SP_a(S) + SP_u(S) < \Big(1 + \frac{m-1}{m}\Big) C_{\max}(S^*).$$

Because $N(S)\ T + SP_a(S) + SP_u(S) = C_{\max}(S)$, we finally have

$$C_{\max}(S) < \Big(1 + \frac{m-1}{m}\Big) C_{\max}(S^*).$$

Case (ii): $SL(S) \leq SP_a(S)$
By Proposition 6, we have

$$N(S)\ T + SP_a(S) + SP_u(S) - SL(S) \leq C_{\max}(S^*). \tag{6}$$

By inequality (2), we have $(m-1)\ SL(S) \leq (m-1)\ N^-(S)\ T$. Since $SP_a(S) \leq (m-1)\ N^+(S)\ T$ (inequality (3)) and $SL(S) \leq SP_a(S)$, we have $SL(S) \leq SP_a(S) \leq (m-1)\ N^+(S)\ T$. Therefore, the sum of these two inequalities yields

$$\begin{aligned} m\ SL(S) &\leq (m-1)\ \big(N^-(S) + N^+(S)\big)\ T \\ &= (m-1)\ N(S)\ T \\ &\leq (m-1)\ N(S^*)\ T \qquad\qquad (\because N(S) \leq N(S^*)) \\ &\leq (m-1)\ C_{\max}(S^*). \end{aligned}$$

Therefore, we have

$$SL(S) \leq \frac{m-1}{m}\ C_{\max}(S^*). \tag{7}$$

By inequalities (6) and (7), we have

$$C_{\max}(S) = N(S)\ T + SP_a(S) + SP_u(S) \leq \Big(1 + \frac{m-1}{m}\Big) C_{\max}(S^*).$$

□

Now, we present a tight instance of the worst case bound. When the number of machines is two, we have

$$C_{\max}(S) \leq \frac{3}{2}\ C_{\max}(S^*).$$

An instance of the tight worst bound for $m = 2$ is illustrated in Figure 5. Jobs A and B have $(T-\epsilon)$ processing time with N units of each required and jobs C and D have ϵ processing time with N units required. In Figure 5 (a) we show the schedule S_a: the first covey consists of jobs A and B for N rotations and the second covey consists of jobs C and D for N rotations. The completion time of schedule S_a is

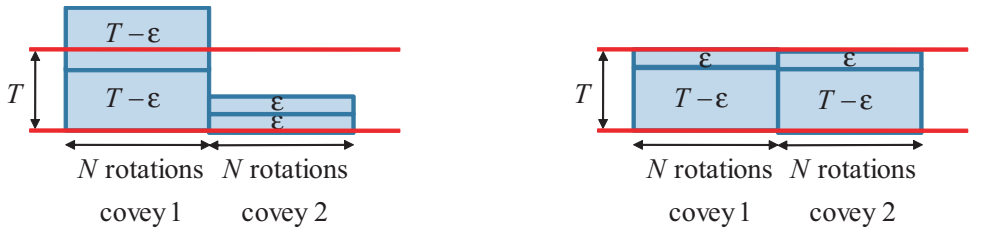
$$\begin{aligned} C_{\max}(S_a) &= N\ \max\{2(T-\epsilon),\ T\} + N\ \max\{2\epsilon,\ T\} \\ &= 2N\ (T-\epsilon) + N\ T \\ &= 3NT - 2N\epsilon. \end{aligned}$$

The schedule S_b illustrated in Figure 5 (b) has jobs A and C in the first covey and jobs B and D

in the second covey. The completion time of schedule S_b is

$$\begin{aligned} C_{\max}(S_b) &= N\ \max\{(T-\epsilon)+\epsilon,\ T\} + N\ \max\{(T-\epsilon)+\epsilon,\ T\} \\ &= 2NT. \end{aligned}$$

Therefore, if ϵ goes to zero, $\frac{C_{\max}(S_a)}{C_{\max}(S_b)} = \frac{3}{2}$, showing that the worst case bound is tight. (It also implies that S_b is an optimal schedule and S_a is a worst-possible schedule.)



(a) the worst schedule with C_{max}= $3NT - 2N\varepsilon$ (b) the optimal schedule with C_{max}= $2NT$

Figure 5: A worst case instance when m=2 and $N(S) \leq N(S^*)$

4.3 Worst Case Bound of the *Longest Unit First* algorithm

In this section, we use results from the previous section to prove a worst case performance bound for the *LUF* algorithm. If schedule $\hat{S}$ produced by the *LUF* algorithm satisfies $N(\hat{S}) \leq N(S^*)$, the worst case bound of Proposition 8 holds for this schedule. Now, consider the opposite case, $N(\hat{S}) > N(S^*)$, when the number of rotations of schedule $\hat{S}$ is greater than that of the optimal schedule.

We partition the rotations of $\hat{S}$ into two subschedules, $\hat{S}_1$ and $\hat{S}_2$, such that $\hat{S}_1$ contains the $N(S^*)$ rotations of $\hat{S}$ with the longest stage 1 processing times, and $\hat{S}_2$ contains the $N(\hat{S}) - N(S^*)$ rotations with the shortest stage 1 processing times.

We define slack time and surplus time for $\hat{S}_1$ and $\hat{S}_2$ as before. Then, the inequalities (2) and (3) still hold for $\hat{S}_1$ and $\hat{S}_2$. Since $\hat{S}_1$ is a subset of $\hat{S}$, we know that the sum of processing times of jobs in schedule $\hat{S}_1$ can be no greater than that in schedule $\hat{S}$, which means that

$$N(\hat{S}_1)\ T + SP_a(\hat{S}_1) + SP_u(\hat{S}_1) - SL(\hat{S}_1)\ \leq\ N(\hat{S})\ T + SP_a(\hat{S}) + SP_u(\hat{S}) - SL(\hat{S}).$$

Therefore, Proposition 6 still holds:

$$N(\hat{S}_1)\ T + SP_a(\hat{S}_1) + SP_u(\hat{S}_1) - SL(\hat{S}_1)\ \leq\ C_{\max}(S^*).$$

Similar to the proof of Proposition 7, we have

$$\sum_{j=1}^{J} n_j \ \max(t_j - T,\ 0) \quad \geq \quad SP_u(\hat{S}) \geq SP_u(\hat{S}_1).$$

In addition, we have

$$\begin{aligned} C_{\max}(S^*) &\geq N(S^*)\ T + \sum_{j=1}^{J} n_j \ \max(t_j - T,\ 0) && (\because \text{Lower Bound 2}) \\ &= N(\hat{S}_1)\ T + \sum_{j=1}^{J} n_j \ \max(t_j - T,\ 0) && (\because N(\hat{S}_1) = N(S^*)) \\ &\geq N(\hat{S}_1)\ T + SP_u(\hat{S}_1). \end{aligned}$$

Therefore, we have a result similar to Proposition 7 for $\hat{S}_1$:

$$C_{\max}(S^*) \quad \geq \quad N(\hat{S}_1)\ T + SP_u(\hat{S}_1).$$

Applying similar arguments as in the proof of Proposition 8, we have

$$C_{\max}(\hat{S}_1) \leq \Big(1 + \frac{m-1}{m}\Big) C_{\max}(S^*). \tag{8}$$

Inequality (8) will be used in the following proof, in which we also consider the remaining rotations $\hat{S}_2$.

Theorem 9. *A schedule $\hat{S}$ produced by the* Longest Unit First *algorithm has a worst case bound of*

$$C_{\max}(\hat{S}) \leq \Big(\ \big(1 + \frac{m-1}{m}\big) \ + \ \big(\frac{1}{3} - \frac{1}{3m}\big) \ \Big) C_{\max}(S^*). \tag{9}$$

Proof. When the number of machines is two, if the number of units of one job is no less than the sum of the number of units of all the other jobs, *LUF* finds a trivial optimal solution with the long job on one machine and all other jobs on the other machine. See Proposition 12 in the Appendix. The proof below excludes this trivial case.

The schedule $\hat{S}$ with $N(\hat{S}) \leq N(S^*)$ produced by *LUF* has a worst case bound of Proposition 8, which satisfies inequality (9). Now, consider schedule $\hat{S}$ with $N(\hat{S}) > N(S^*)$, and define $\hat{S}_1$ and $\hat{S}_2$ as above.

In the parallel machine scheduling problem, the *Longest Processing Time First (LPT)* algorithm

has the following worst case bound (Graham (1969)):

$$C_{\max}(LPT) \leq \left(\frac{4}{3} - \frac{1}{3m} \right) C_{\max}(OPT)$$

where LPT and OPT represent the schedule produced by the LPT algorithm and the schedule of an optimal solution, respectively. The *Longest Unit First (LUF)* algorithm of FGSP is similar to LPT and, we can apply the Graham theorem to the number of rotations for $\hat{S}_1, \hat{S}_2$ and S^* of LUF. We already observed that the unit model is equivalent to the parallel machine scheduling problem. By their relation, the makespan of LPT in the parallel machine scheduling problem, $C_{\max}(LPT)$, corresponds to the number of rotations $N(\hat{S})$ for schedule $\hat{S}$, and $C_{\max}(OPT)$ corresponds to $N(S^*)$. Hence, since $N(\hat{S}_1) = N(S^*)$, we have

$$\begin{aligned} N(\hat{S}_1) + N(\hat{S}_2) &\leq \left(\frac{4}{3} - \frac{1}{3m}\right) N(S^*) \\ N(\hat{S}_2) &\leq \left(\frac{1}{3} - \frac{1}{3m}\right) N(S^*). \end{aligned} \tag{10}$$

Case (i): If the sum of times for each rotation of $\hat{S}_2$ is no greater than T, we have

$$\begin{aligned} C_{\max}(\hat{S}_2) &= N(\hat{S}_2)\ T \\ &\leq (\frac{1}{3} - \frac{1}{3m}) N(S^*)\ T \qquad \text{(by (10))} \\ &\leq (\frac{1}{3} - \frac{1}{3m}) C_{\max}(S^*). \end{aligned}$$

Therefore, we have

$$\begin{aligned} C_{\max}(\hat{S}) = C_{\max}(\hat{S}_1) + C_{\max}(\hat{S}_2) &\leq (1 + \frac{m-1}{m}) C_{\max}(S^*) \ + \ (\frac{1}{3} - \frac{1}{3m}) C_{\max}(S^*) \\ &= \left((1 + \frac{m-1}{m}) \ + \ (\frac{1}{3} - \frac{1}{3m}) \right) C_{\max}(S^*). \end{aligned}$$

Case (ii): Suppose that at least one rotation in $\hat{S}_2$ has total stage 1 processing time greater than T. Let $t_{\max}$ be the maximum sum of times among rotations in schedule $\hat{S}_2$, that is, $t_{\max} := \max_{C_i \in \hat{S}_2} t(C_i)$. Since we take $\hat{S}_2$ from $\hat{S}$ so that the sum of times for each covey in $\hat{S}_2$ is as small as possible, the total stage 1 processing time of coveys with a rotation in $\hat{S}_1$ should be no less than

$t_{\max}$. Therefore, we have

$$\begin{aligned}
C_{\max}(\hat{S}_2) &\leq N(\hat{S}_2)\ t_{\max} && (\because t(C_k) \leq t_{\max},\ \forall C_k \in \hat{S}_2\ ) \\
&\leq \big(\frac{1}{3} - \frac{1}{3m}\big)\ N(\hat{S}_1)\ t_{\max} && (\because \text{inequality (10) }) \\
&\leq \big(\frac{1}{3} - \frac{1}{3m}\big)\ \sum_{C_k \in \hat{S}_1} n(C_k)\ t(C_k) && (\because t(C_k) \geq t_{\max},\ \forall C_k \in \hat{S}_1\ ) \\
&\leq \big(\frac{1}{3} - \frac{1}{3m}\big)\ \sum_{C_k \in \hat{S}} n(C_k)\ t(C_k) \\
&= \big(\frac{1}{3} - \frac{1}{3m}\big)\ \sum_{j \in J} t_j\ n_j \\
&\leq \big(\frac{1}{3} - \frac{1}{3m}\big)\ C_{\max}(S^*)
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
C_{\max}(\hat{S}) = C_{\max}(\hat{S}_1) + C_{\max}(\hat{S}_2) &\leq \big(1 + \frac{m-1}{m}\big) C_{\max}(S^*)\ +\ \big(\frac{1}{3} - \frac{1}{3m}\big) C_{\max}(S^*) \\
&= \Big(\ \big(1 + \frac{m-1}{m}\big)\ +\ \big(\frac{1}{3} - \frac{1}{3m}\big)\ \Big) C_{\max}(S^*).
\end{aligned}$$

□

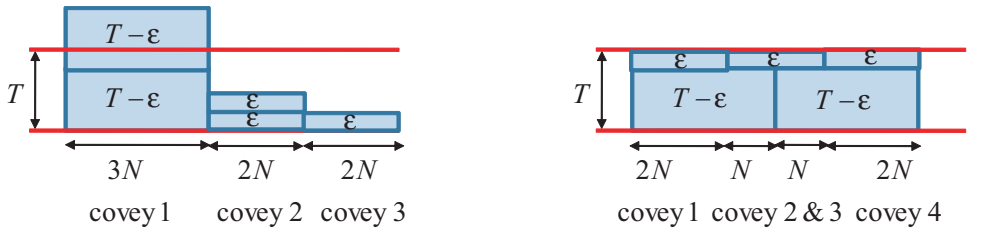
Now, we present an instance showing that the worst case bound is tight. When the number of machines is two, the theorem states

$$C_{\max}(\hat{S}) \leq \frac{10}{6}\ C_{\max}(S^*).$$

An instance of the tight worst bound for $m = 2$ is illustrated in Figure 6. Jobs A and B have $(T - \epsilon)$ processing time with $3N$ units of each required and jobs C, D and F have ϵ processing time with $2N$ units required. Schedule S_a of Figure 6 (a): the first covey consists of jobs A and B for $3N$ rotations, the second covey consists of jobs C and D for $2N$ rotations, and the third covey consists of only job E for $2N$ rotations. The completion time of schedule S_a is

$$\begin{aligned}
C_{\max}(S_a) &= 3N\ \max\{2\ (T - \epsilon),\ T\} + 2N\ \max\{2\ \epsilon,\ T\} + 2N\ \max\{\epsilon,\ T\} \\
&= 6N\ (T - \epsilon) + 2N\ T + 2N\ T \\
&= 10NT - 6N\epsilon.
\end{aligned}$$

In the schedule S_b illustrated in Figure 6 (b), the first machine produces job A and then B, and the second machine produces job D, E and then F. Then, there are four coveys and the completion



(a) the worst schedule with $C_{max}= 10NT - 6N\varepsilon$ (b) the optimal schedule with $C_{max}= 6NT$

Figure 6: A worst case instance of the *Longest Unit First* when m=2

time of schedule S_b is

$$\begin{aligned} C_{\max}(S_b) &= 2N\ \max\{(T-\epsilon)+\epsilon,\ T\} + N\ \max\{(T-\epsilon)+\epsilon,\ T\} \\ &\quad +N\ \max\{(T-\epsilon)+\epsilon,\ T\} + 2N\ \max\{(T-\epsilon)+\epsilon,\ T\} \\ &= 6NT. \end{aligned}$$

Therefore, if ϵ goes to zero, $\frac{C_{\max}(S_a)}{C_{\max}(S_b)} = \frac{10}{6}$, showing that the worst case bound is tight. (It also implies that S_b is an optimal schedule and S_a is a worst-possible schedule.)

References

Birewar, D.B., and Grossmann, I.E. (1989-I) Incorporating Scheduling in the Optimal Design of Multiproduct Batch Plants *Computers and Chemical Engineering*, **13**, 141–161.

Birewar, D.B., and Grossmann, I.E. (1989-II) Efficient Optimization Algorithms for Zero-Wait Scheduling of Multiproduct Batch Plants *Industrial and Engineering Chemistry Research*, **28**, 1333–1345.

Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Co.

Graham, R.L. (1969) Bounds on Multiprocessing Timing Anomalies. *SIAM Journal of Applied Mathematics*, **17**, 263–269.

Gupta, J. N. D. and Tunc, E. A. (1991) Schedules for a Two Stage Hybrid Flowshop with Parallel Machines at the Second Stage. *Int. J. Prod. Res.*, **29**, 1489–1502.

Linn, R. and Zhang, W. (1999) Hybrid Flow Shop Scheduling: A Survey. *Computers and Industrial Engineering*, **37**, 57–61.

McCormick, S.T., Pinedo, M.L., Shenker, S., and Wolf, B. (1989) Sequencing In an Assembly Line with Blocking to Minimize Cycle Time. *Operations Research*, **37**, 925–935.

Na, B., Ahmed, S., Nemhauser, G. L., and Sokol, J. (2012) Optimization of Automated Float Glass Lines in review at *International Journal of Production Economics*.

Pinedo, M.L. (2008) *Scheduling: Theory, Algorithms, and Systems*. 3rd Edition Springer.

Sriskandarajah, C. (1993) Performance of Scheduling Algorithms for No-wait flowshops with Parallel Machines *European Journal of Operational Research*, **70**, 365–378.

Appendix

A Relation between a Mixed Covey and Minimal Coveys

In this section of the appendix, we provide a rule for dividing a mixed covey into minimal coveys, and prove that the process does not increase the amount of scrap needed in the solution.

Recall that a covey in which each job appears at most once is referred to as a *minimal covey*, and a covey in which at least one job appears more than once is called a *mixed covey*. For example, the covey $\{a, x, y, b, z, y, z, x, c\}$ is a mixed covey because jobs x, y, and z appear more than once in the covey. We refer to such jobs that appear more than once in a covey as the covey's *duplicating jobs*.

By the machine dedication restriction, all instances of a duplicating job in a mixed covey must be assigned to the same offloading machine. Therefore, within a mixed covey, the time between any two consecutive cuts of the same job (including wrapping around from the end of the covey to the start) must be at least as long as the cycle time of the offloading machine.

It is more convenient to analyze FGSP with minimal coveys than with mixed coveys, but it is important to ensure that restricting the solution space to minimal coveys does not increase the amount of scrap required. The following algorithm for creating minimal coveys from a mixed covey guarantees that scrap will not need to increase.

Algorithm: Mixed-to-Minimal Coveys

(Step 0) Begin with a mixed covey M that is required to be run K times. Create the sequence S of jobs that will be cut in those K runs of mixed covey M. (So, if M has J jobs (including duplicates), then S will be a sequence of $J \times K$ jobs.)

(Step 1) From the beginning of S, put the jobs into a new minimal covey in the same order as in S, until a duplicate is found or the end of S is reached. Remove those jobs from S. If S is still nonempty, repeat Step 1. The first and last iterations of Step 1 might yield unique coveys, but all other iterations will yield $K - 1$ duplicates of the same coveys.

(Step 2) Re-order the minimal coveys so that identical ones are grouped (and now call them a single covey that is run more than once).

For example, consider the covey $\{a, x, y, x, b, y\}$ that is to be run K times. Step 0 creates the sequence $\{a, x, y, x, b, y, a, x, y, x, b, y, a, x, y, x, b, y, \ldots, a, x, y, x, b, y\}$. The first iteration of Step 1 finds minimal covey $\{a, x, y\}$ before a duplicate job (x) is found. The remaining sequence is $\{x, b, y, a, x, y, x, b, y, a, x, y, x, b, y, \ldots, a, x, y, x, b, y\}$. The second iteration of Step 1 finds minimal covey $\{x, b, y, a\}$ before duplicate job x is found, and the remaining sequence is $\{x, y, x, b, y, a, x, y, x, b, y, \ldots, a, x, y, x, b, y\}$. Iterations of Step 1 continue to alternate between finding minimal covey $\{x, y\}$ and minimal covey $\{x, b, y, a\}$, until the last iteration when only $\{x, b, y\}$ remains in the sequence.

So, at the end of Step 1, the set of minimal coveys, each to be run once, is:
$\{a, x, y\}$, $\{x, b, y, a\}$, $\{x, y\}$, $\{x, b, y, a\}$, $\{x, y\}$, ..., $\{x, b, y, a\}$, $\{x, y\}$, $\{x, b, y\}$.

In Step 2, we group similar coveys to get the final minimal-covey solution:
$\{a, x, y\}_{(1 \text{ run})}$
$\{x, b, y, a\}_{(K-1 \text{ runs})}$
$\{x, y\}_{(K-1 \text{ runs})}$
$\{x, b, y\}_{(1 \text{ run})}$.

A covey of which the number of rotation (run) is one is referred to as a *singleton covey.* In the above example, $\{a, x, y\}$ is the first singleton covey and $\{x, b, y\}$ is the last singleton covey.

Proposition 10. *Excluding transient scrap between coveys and the first and last singleton coveys if applicable, the amount of scrap required by the minimal covey solution created by the Algorithm Mixed-to-Minimal Coveys is no greater than the scrap required by the corresponding mixed covey solution.*

Proof. Let T be the offloading machine cycle time. Consider any covey C that is not a first or last singleton covey in the solution created by the Algorithm Mixed-to-Minimal Coveys. Denote the jobs of a run of this covey as $j_1, \ldots, j_{|C|}$. Let j' be the job immediately following these jobs of the run of C in the original mixed covey.

By Step 1 of the Algorithm Mixed-to-Minimal Coveys, j' must be a duplicate of a job in C; otherwise, the algorithm would have included j' in C. Without loss of generality, suppose j' is a duplicate of the kth job in C (i.e., job j_k). In the original mixed covey solution, j' and the jobs of C must have appeared in the order $j_1, \ldots, j_{|C|}, j'$. Therefore, between duplicate jobs j_k and j', the mixed covey must have incurred at least $\max\{0,\ T - (t_k + \cdots + t_{|C|})\} = S_{\text{mixed}}$ scrap.

On the other hand, the minimal covey C has total cutting time $t_1 + \cdots + t_{|C|}$, so C incurs scrap equal to $S_{\text{minimal}} = \max\{0,\ T - (t_1 + \cdots + t_{|C|})\}$. Since $k \geq 1$, it must be that $S_{\text{minimal}} \leq S_{\text{mixed}}$.

So, the scrap incurred by a run of minimal covey C is no more than the scrap incurred between the same set of jobs in the original mixed covey. Since the minimal coveys exactly partition the jobs of

the original mixed covey (other than the first and last transient singletons), and the jobs considered when calculating each S_{mixed} are a subset of the jobs in the corresponding minimal covey, it must therefore be true that the total scrap incurred by the repeating minimal coveys is no more than the total scrap incurred between those same jobs in the original mixed covey. □

B Match Largest and Smallest Jobs

Proposition 11. *The algorithm* match largest and smallest jobs *produces an optimal solution in the time model when the number of machines is two.*

Proof. We prove the proposition for an even number $2k$ of jobs; for an odd number of jobs, we can reduce to the even case by adding a dummy job with $t = 0$. We first prove that an optimal solution exists where every covey contains two jobs, and then show that the algorithm produces a solution that is at least as good as any other solution with two jobs in each covey, and is thus optimal overall.

To prove that an optimal solution exists where each covey has exactly two jobs, consider an optimal solution S_1 with a covey C_p that consists of one job p with processing time t_p. Since the number of jobs is even, there exists another covey $C_q \in S_1$ that also consists of one job q with processing time t_q. We can easily construct another solution S_2 that is identical to S_1 except that coveys C_p and C_q are replaced by a single covey C_r containing jobs p and q. The completion time of S_2 is no greater than that of S_1:

$$C_{\max}(S_1) - C_{\max}(S_2) \quad = \quad \big\{ \max\{T,\ t_p\} + \max\{T,\ t_q\} \big\} - \max\{T,\ t_p + t_q\} \geq 0.$$

Without loss of generality, assume that the jobs $j_1, \ldots, j_{2k}$ have stage 1 processing times $t_1 \leq t_2 \leq \cdots \leq t_{2k}$. Then, applying the algorithm yields a solution with following coveys: $\{j_1, j_{2k}\}, \{j_2, j_{2k-1}\}, \cdots, \{j_k, j_{k+1}\}$. Each covey consists of two jobs, and the jobs j_i and j_{2k-i+1} are matched.

Consider a solution $S_2 \neq S^*$ with two jobs in each covey, such that the completion time of S_2 is at least as small as that of any other solution with two jobs per covey.

We can transform S_2 into S^* using the following k-step exchange procedure. We begin with S_2. At each step i of the procedure, for $1 \leq i \leq k$, if jobs j_i and j_{2k-i+1} are in the same covey, we leave the solution unchanged. Otherwise, there must be jobs j_{u_i} and j_{v_i} such that the solution contains coveys $\{j_i, j_{u_i}\}$ and $\{j_{v_i}, j_{2k-i+1}\}$. In this case, modify the solution by replacing coveys $\{j_i, j_{u_i}\}$ and $\{j_{v_i}, j_{2k-i+1}\}$ with coveys $\{j_i, j_{2k-i+1}\}$ and $\{j_{u_i}, j_{v_i}\}$. After k steps, we will be left with solution S^*. We note that for each step i where a modification is made, $i < u_i, v_i < 2k - i + 1$ since the procedure guarantees all jobs smaller than i and larger than $2k - i + 1$ will already be matched as in solution S^*. Thus, $t_i < t_{u_i}, t_{v_i} < t_{2k-i+1}$.

Below, we prove that no step of the exchange procedure will increase the completion time. Therefore, $C_{\max}(S^*) \leq C_{\max}(S_2)$, so S^* must be an optimal solution.

Claim: No step of the exchange procedure increases completion time.

Proof of Claim: If no modification is made, there is no change in completion time. So, consider a step where coveys $\{j_i, j_{u_i}\}$ and $\{j_{v_i}, j_{2k-i+1}\}$ are replaced by coveys $\{j_i, j_{2k-i+1}\}$ and $\{j_{u_i}, j_{v_i}\}$. We show the proof when $t_i + t_{2k-i+1} \leq t_{u_i} + t_{v_i}$; the proof for $t_i + t_{2k-i+1} > t_{u_i} + t_{v_i}$ is similar.

Case 1: $t_{u_i} \leq t_{v_i}$

For the following subcases, we will prove that the completion time of coveys $\{j_i, j_{u_i}\}$ and $\{j_{v_i}, j_{2k-i+1}\}$ is no greater than that of coveys $\{j_i, j_{2k-i+1}\}$ and $\{j_{u_i}, j_{v_i}\}$; i.e. $\max\{T,\ t_i + t_{u_i}\} + \max\{T,\ t_{v_i} + t_{2k-i+1}\} \geq \max\{T,\ t_i + t_{2k-i+1}\} + \max\{T,\ t_{u_i} + t_{v_i}\}$.

Subcase 1.1: $t_{v_i} + t_{2k-i+1} \leq T$

$$\begin{aligned} \max\{T, t_i + t_{u_i}\} + \max\{T, t_{v_i} + t_{2k-i+1}\} &= T + T \\ &= \max\{T, t_i + t_{2k-i+1}\} + \max\{T, t_{u_i} + t_{v_i}\} \end{aligned}$$

Subcase 1.2: $t_i + t_{2k-i+1} < T \leq t_{v_i} + t_{2k-i+1}$

$$\begin{aligned} \max\{T, t_i + t_{u_i}\} + \max\{T, t_{v_i} + t_{2k-i+1}\} &= T + \max\{T, t_{v_i} + t_{2k-i+1}\} \\ &\geq T + \max\{T, t_{u_i} + t_{v_i}\} \\ &\quad (\because \max\{T, t_{v_i} + t_{2k-i+1}\} \geq \max\{T, t_{v_i} + t_{u_i}\}) \\ &= \max\{T, t_i + t_{2k-i+1}\} + \max\{T, t_{u_i} + t_{v_i}\} \end{aligned}$$

Subcase 1.3: $T \leq t_i + t_{2k-i+1}$

$$\begin{aligned} \max\{T, t_i + t_{u_i}\} + \max\{T, t_{v_i} + t_{2k-i+1}\} &= \max\{T, t_i + t_{u_i}\} + t_{v_i} + t_{2k-i+1} \\ &\geq (t_i + t_{u_i}) + (t_{v_i} + t_{2k-i+1}) \\ &= \max\{T, t_i + t_{2k-i+1}\} + \max\{T, t_{u_i} + t_{v_i}\} \end{aligned}$$

Case 2: $t_{u_i} > t_{v_i}$

Case 2 can be proved similarly to Case 1. □

C Trivial Solution for FGSP

FGSP has a trivial optimal solution in the following case when the number of machines is two.

Proposition 12. *When the number of machines is two, if the number of units of one job (the*

"long job") is no less than the sum of the number of units of all the other jobs, an optimal schedule of FGSP is for every covey to include the long job. That is, all other jobs are processed on one machine while the long job is simultaneously processed on the other machine. This characterizes all optimal solutions if no job's unit processing time is T or greater.

Proof. (proof by contradiction) Assume that in the optimal schedule there exists a covey that does not include the long job, which we arbitrarily refer to as job J_0. There are two cases: (i) there exists a one-job covey with job $J_1 \neq J_0$, and (ii) there exists a two-job covey $\{J_1, J_2\}$ where $J_1 \neq J_0$ and $J_2 \neq J_0$. (A solution might also satisfy both cases.)

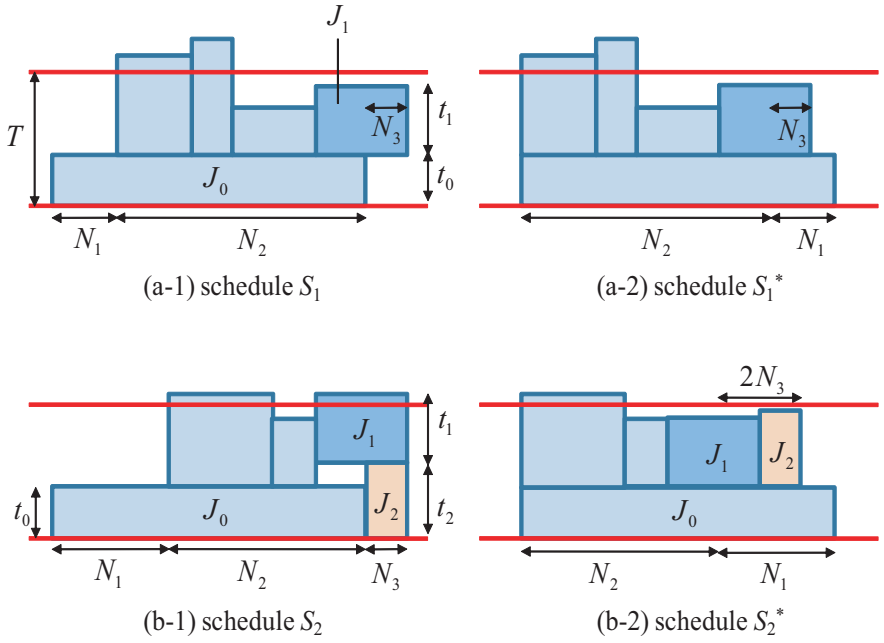


Figure 7: Trivial optimal schedule of FGSP when the number of machines is two.

Figure 7 (a-1) shows such a case, and Figure 7 (a-2) shows the intuition of how to create an improved solution S_1^* by shifting all the jobs on the second machine to eliminate the one-job covey that does not include J_0. As shown below, $C_{\max}(S_1) \geq C_{\max}(S_1^*)$, with strict improvement unless jobs J_0 and J_1 each require at least T processing time by themselves.

$$
\begin{aligned}
C_{\max}(S_1) &= N_1 \max(t_0, T) + N_3 \max(t_1, T) + C_{\max}(N_2 \text{ area}) \\
&= (N_1 - N_3) \max(t_0, T) + N_3 \max(t_0, T) + N_3 \max(t_1, T) + C_{\max}(N_2 \text{ area}) \\
&\geq (N_1 - N_3) \max(t_0, T) + N_3 \max(t_0 + t_1,\ T) + C_{\max}(N_2 \text{ area}) \\
&\qquad\qquad (\because \max(A, T) + \max(B, T) \geq \max(A + B,\ T)) \\
&= C_{\max}(S_1^*)
\end{aligned}
$$

Figures 7 (b-1) and 7 (b-2) show similar intuition for the second case.

$$\begin{aligned}
C_{\max}(S_2) &= N_1 \max(t_0, T) + N_3 \max(t_1 + t_2,\ T) + C_{\max}(N_2 \text{ area}) \\
&= (N_1 - 2N_3) \max(t_0, T) + 2N_3 \max(t_0, T) + N_3 \max(t_1 + t_2,\ T) + C_{\max}(N_2 \text{ area}) \\
&\geq (N_1 - 2N_3) \max(t_0, T) + N_3 \max(t_0 + t_1,\ T) + N_3 \max(t_0 + t_2,\ T) + C_{\max}(N_2 \text{ area}) \\
&\quad \text{(by Lemma 13: } 2\max(t_0, T) + \max(t_1 + t_2,\ T) \geq \max(t_1 + t_0,\ T) + \max(t_2 + t_0,\ T)) \\
&= C_{\max}(S_2^*)
\end{aligned}$$

Therefore, in an optimal schedule, all coveys should have the long job, J_0. □

Lemma 13. *When t_0, t_1, t_2, and $T \geq 0$, we have*

$$\max(t_1 + t_0,\ T) + \max(t_2 + t_0,\ T) \leq 2\max(t_0, T) + \max(t_1 + t_2,\ T).$$

Proof. Without loss of generality, assume that $t_1 \geq t_2$. Then, we can consider three cases.

Case 1: $t_0 \geq t_1 \geq t_2$

$$\begin{aligned}
& \max(t_1 + t_0,\ T) + \max(t_2 + t_0,\ T) \\
= \ & \max(t_1 + t_0 - t_1,\ T - t_1) + t_1 + \max\big(t_2 + t_0 - (t_0 - t_1),\ T - (t_0 - t_1)\big) + (t_0 - t_1) \\
& \qquad \big(\because t_1 \geq 0,\ (t_0 - t_1) \geq 0\ \big) \\
\leq \ & \max(t_0,\ T) + t_0 + \max(t_1 + t_2,\ T) \\
& \qquad \big(\because \max(t_0,\ T - t_1) \leq \max(t_0,\ T), \quad \max\big(t_1 + t_2,\ T - (t_0 - t_1)\big) \leq \max(t_1 + t_2,\ T)\ \big) \\
\leq \ & 2\max(t_0,\ T) + \max(t_1 + t_2,\ T) \qquad \big(\because t_0 \leq \max(t_0,\ T)\ \big)
\end{aligned}$$

Case 2: $t_1 \geq t_0 \geq t_2$

$$\begin{aligned}
& \max(t_1 + t_0,\ T) + \max(t_2 + t_0,\ T) \\
= \ & \max\big(t_1 + t_0 - (t_0 - t_2),\ T - (t_0 - t_2)\big) + (t_0 - t_2) + \max(t_2 + t_0 - t_2,\ T - t_2) + t_2 \\
& \qquad \big(\because (t_0 - t_2) \geq 0,\ t_2 \geq 0\ \big) \\
\leq \ & \max(t_1 + t_2,\ T) + t_0 + \max(t_0,\ T) \\
& \qquad \big(\because \max\big(t_1 + t_2,\ T - (t_0 - t_2)\big) \leq \max(t_1 + t_2,\ T), \quad \max(t_0,\ T - t_2) \leq \max(t_0,\ T)\ \big) \\
\leq \ & \max(t_1 + t_2,\ T) + 2\max(t_0,\ T) \qquad \big(\because t_0 \leq \max(t_0,\ T)\ \big)
\end{aligned}$$

Case 3: $t_1 \geq t_2 \geq t_0$

$$
\begin{aligned}
& \max(t_1 + t_0,\ T) + \max(t_2 + t_0,\ T) \\
= \ & \max(t_1 + t_0 - t_1,\ T - t_1) + t_1 + \max(t_2 + t_0 - t_2,\ T - t_2) + t_2 \qquad (\because t_1 \geq 0,\ t_2 \geq 0\ ) \\
\leq \ & \max(t_0,\ T) + \max(t_0,\ T) + t_1 + t_2 \\
& \qquad \big(\ \because \max(t_0,\ T - t_1) \leq \max(t_0,\ T), \quad \max(t_0,\ T - t_2) \leq \max(t_0,\ T)\ \big) \\
\leq \ & 2\max(t_0,\ T) + \max(t_1 + t_2,\ T) \qquad \big(\ \because t_1 + t_2 \leq \max(t_1 + t_2,\ T)\ \big)
\end{aligned}
$$

Therefore, we have $\max(t_1 + t_0,\ T) + \max(t_2 + t_0,\ T) \leq 2\max(t_0, T) + \max(t_1 + t_2,\ T)$. $\square$