Technical Memorandum Number 736

The Use of Flowlines to Simplify Routing Complexity in Two-Stage Flowshops

by

George Vairaktarakis
Mohsen Elhafsi

August 2000

Department of Operations
Weatherhead School of Management
Case Western Reserve University
10900 Euclid Avenue
Cleveland, Ohio 44106-7235

The Use of Flowlines to Simplify Routing Complexity in Two-stage Flowshops

George Vairaktarakis *   Mohsen Elhafsi †

IIE Transactions, 32(8):687-699, 2000

Abstract

Flexible manufacturing systems are often designed as flowshops supported by automated material handling devices that facilitate routing among any two processors of adjacent stages. This routing structure is complex, and results in excessive capital investment and costs of management. In this paper we propose a decomposition of two stage flowshops into smaller independent flowlines that allow for unidirectional routing only. We solve optimally the problem of minimizing makespan on 2 parallel flowlines, by means of a dynamic programming algorithm (DP). Based on DP we develop lower bounds on the throughput performance of environments that consist of more than two flowlines. We present several heuristic algorithms and report their optimality gaps. Using these algorithms, we show that the decomposition of two stage flowshops with complicated routing into flowline-like designs with unidirectional routing is associated with minor losses in throughput performance, and hence significant savings in material handling costs.

Keywords: Hybrid Flowshops, Scheduling, Routing control, Dynamic programming.

*Weatherhead School of Management, Dept. of Operations, Case Western Reserve University, Cleveland, OH 44106-7235

†The A. Gary Anderson Graduate School of Management, University of California, Riverside, CA 92521-0203

1 Introduction

Flexible manufacturing systems (FMS) coupled with cellular manufacturing is the preferred way of producing in medium to large volumes (see Maleki, 1991). The study of scheduling problems in flexible manufacturing systems has attracted significant attention in recent years including Afentakis, 1986, Erschler *et al.*, 1985, Ghosh and Gaimon, 1992, Kouvelis and Vairaktarakis, 1998, Lee and Vairaktarakis, 1998, Shanker and Tzen, 1985, Stecke, 1985, 1992, and Wittrock, 1988, due to the importance of such systems for small-to-medium batch manufacturing.

In many cases, the production system consists of several manufacturing cells each of which is structured as a multistation flowshop (see Baker, 1993, Blakewicz, 1993, and Pinedo, 1995). A typical design for such cells is a multistage system where each stage consists of multiple identical machines. Among the most popular flowshop designs is the *hybrid flowshop* which we denote by HFS_{m_1,m_2} (see Figure 1). It consists of m_1 machines in stage 1, and m_2 machines in stage 2. Each job consists of two tasks a_i and b_i that are to be processed in this order in stages 1 and 2 respectively. The a_i task can be processed on any of the m_1 machines of stage 1, and the b_i task can be processed on any of the m_2 machines of stage 2. As a result, the HFS_{m_1,m_2} system enjoys routing flexibility.

The above descibed HFS system results in high throughput rates. These rates come at the expense of sophisticated material handling systems that consist of a combination of automated guided vehicles and automated transfer lines. Such handling systems usually require major investment both in capital and management technology. In this paper we present a way of decomposing the HFS_{m_1,m_2} design into smaller independent cells that require unidirectional routing. The proposed design results to substantial reduction in routing complexity (and hence savings in material handling costs), and simplified management of the production system. Without loss of generality we assume that $m_1 \leq m_2$ since the opposite case is symmetric. More specifically, we decompose HFS_{m_1,m_2} into m_1 independent units each of which is a hybrid flowshop of the form $HFS_{1,k}$ where k is an appropriate integer (see Figure 1). This decomposition of HFS_{m_1,m_2} has the advantage of forcing unidirectional routing from the only stage 1 machine to the k stage 2 machines. We refer to this design as *Parallel* HFS because it consists of several $HFS_{1,k}$'s operated in parallel. To the best of our knowledge, the PHFS design has not being studied in the literature before. Our analysis shows that the scheduling task within each $HFS_{1,k}$ sub-design is significantly simpler and more accurate than the corresponding task in HFS_{m_1,m_2}. In addition, an allocation of jobs to the $HFS_{1,k}$ cells allows to manage each cell independently thus focusing in on-time completion of a smaller subset of jobs. In contrast, a delay on any machine of HFS_{m_1,m_2} may affect the throughput performance of the entire production system.

In a nutshell, this article considers the following managerial question: *How significant is the deterioration in the makespan performance of PHFS (where the routing structure is the simplest possible) as compared to the makespan performance of HFS_{m_1,m_2} (where the routing structure is as complicated as possible for a 2-stage system)?* As shown in Section 5, the answer is that in the majority of cases that we experimented with, the deterioration of the makespan performance of PHFS is less than 3%. This conclusion can have significant impact on the process design used to implement 2-stage flowshop production systems.

In order to present the decomposition of HFS_{m_1,m_2} into the PHFS system, we first study a simpler but equally important design; the m parallel flowlines design denoted by mFL (see Figure 1). In the next section we formally define the mFL design, review the existing literature on the HFS_{m_1,m_2} and mFL systems, and provide an outline of the rest of the paper.

2 Problem Description and Literature Review

The mFL design (see Figure 1) and the associated makespan problem are defined as follows. Let J be a given set of jobs, where $J_i = (a_i, b_i)$ for $1 \leq i \leq n$. The jobs in J are to be processed by a system of m parallel flowlines $L_1, L_2, \ldots, L_m$ where each flowline is a 2-machine flowshop (see Johnson, 1954), as in Figure 1. Our problem is to assign the jobs in J to the m flowlines, and then schedule the jobs assigned to each flowline so as to minimize makespan. Due to the fact that Johnson's algorithm is optimal for the 2-machine flowshop, the core of our problem is to identify an optimal assignment of jobs in J to the m flowlines. We refer to the above design and protocol of operations as the mFL problem. The mFL design is a special case of PHFS, and has first appeared in He *et al.*, 1996. In this, the authors consider a design that consists of several flowlines (i.e., traditional flowshops with a single processor per stage) operated in parallel, motivated by an application from the glass industry. They consider several product types, setup times between different types, and no-wait in process. The latter constraint renders this problem very different than mFL.

The mFL problem offers an alternative design for flexible flowshops with routing flexibility. In mFL, the production system is decomposed into m independent units or cells, each of which can be managed independently. This approach is aligned with the principles of cellular manufacturing that have gained polularity over the last few years. Admittedly, the mFL design will incur a loss in throughput performance as compared to an equivalent design that allows free routing between stages. In this paper we quantify this throughput loss and assess the benefits of the simplified routing control structure of mFL.

Note that when $m_1 = m_2 = m$, the HFS_{m_1,m_2} system has the same number and layout of machines as the mFL design. These two systems differ only in the routing control structure. Both the HFS and mFL designs are generalizations of the m parallel identical machine environment (mP) where it is assumed that $b_i = 0$ for every $J_i \in J$. The mP environment has been well studied over the last 25 years by several researchers. A review including most results in this area is given by Cheng and Sin, 1990. Garey and Johnson, 1979, have shown that minimizing makespan in the mP environment is ordinary $\mathcal{NP}$-complete.

A great amount of research has been devoted to the HFS problem to minimize makespan. A survey of articles on the problem of minimizing makespan in HFS_{m_1,m_2} that appeared prior to 1993 is provided in Lee and Vairaktarakis, 1994 with significant detail. Also, the authors present a heuristic algorithm for HFS_{m_1,m_2}, which has near optimal performance for randomly generated problems (average relative gaps are less than 1%), and a worst case error bound of $1 - \frac{1}{\max\{m_1,m_2\}}$. This bound appears to be the best known bound for HFS_{m_1,m_2} and its sub-design $HFS_{m,1}$. Since 1993, the papers that appeared in the literature include the following. Hoogeveen *et al.*, 1996 prove that $HFS_{2,1}$ is strongly NP-complete (a problem that was open since the inception of the HFS_{m_1,m_2} problem) thus settling the complexity of HFS_{m_1,m_2} completely. Guinet and Solomon, 1996 compare the performance of several heuristics on hybrid flowshops with 3 or 5 stages. For the makespan objective they find that the best among the heuristics considered in their study exhibits a relative deviation (from the lower bound) of about 8% on the average. They also consider the performance of these heuristics in minimizing maximum tardiness. Gupta *et al.*, 1997 develop a branch and bound algorithm for $HFS_{m,1}$, and present computational results on problems with up to 20 jobs within 25-30 seconds on an IBM 3090 computer. Solomon *et al.*, 1996 present 5 heuristics, and experiment with problems of size 50, 150, and 300 jobs. The reported relative deviations from their lower bounds range from 0.75% to 4.27% (on the average) depending on the heuristics, and the range of the task processing times.

Before we proceed with the analysis of the mFL problem, we present a basic heuristic algorithm (LV) for minimizing the makespan of HFS_{m_1,m_2} (see Lee and Vairaktarakis, 1994). This algorithm will be used later to decompose the HFS_{m_1,m_2} design into a PHFS design which is a parallel implementation of flowline-like units that support unidirectional routing (see Figure 1). The LV heuristic utilizes the *first available machine* rule (FAM). In this, the job to be scheduled next on m parallel identical machines, is assigned to the first machine that becomes available i.e., the machine that finishes first the job (if any) previously assigned to it. Depending on the starting order S, the FAM rule produces different solutions. Also, the LV heuristic utilizes the *last busy machine* rule (LBM), which is the mirror image of FAM. The LBM rule is described below for a given constant $T > 0$ and an ordering S of tasks $\{c_i : 1 \leq i \leq n\}$.

<u>LBM rule:</u>

1. Set $t_k := T$ for $k = 1, \ldots, m_2$.

2. Let c_i be the last unscheduled task of S and M_k a machine with largest t_k. Schedule the task c_i on the machine M_k to finish at time t_k.

3. Set $t_k := t_k - c_i$ and $S := S - \{c_i\}$. If $S \neq \emptyset$ then goto 2 else Stop.

In the above rule, the value of t_k is the time that the machine M_k becomes busy. In step 2 we assign the task c_i to a machine with largest t_k, i.e. the last busy machine. Hence, we call this rule the last busy machine rule. Also, note that the value of T is only a reference point and has no effect on the allocation of tasks to machines. With this background we can present the LV algorithm.

LV algorithm

1. Apply the Johnson's algorithm with respect to the processing times $\{(\frac{1}{m_1}a_i, \frac{1}{m_2}b_i) : i = 1, 2, \ldots, n\}$. Let S be the resulting sequence.

2. Apply the FAM rule on the stage 1 tasks of the sequence S

3. Apply the LBM rule on the stage 2 tasks of the sequence S

4. On each stage 2 machine M_{k2}, reorder the tasks assigned during step 3 so that no task appears before another task that has a smaller completion time at stage 1. Let S_k be the resulting order for $k = 1, 2, \ldots, m_2$.

5. On each stage 2 machine M_{k2}, schedule the tasks in S_k in this order, as soon as possible.

At step 1 of LV a sequence S of jobs is produced, at step 2 an assignment of a_i tasks to the stage 1 machines is made and at steps 3,4 and 5 the tasks of stage 2 are scheduled. In particular, step 3 determines which tasks will be processed by each stage 2 machine, step 4 determines the order of stage 2 tasks within a stage 2 machine, and step 5 proceeds with the scheduling of the stage 2 tasks on stage 2 machines. Since Johnson's algorithm requires $\mathcal{O}(n \log n)$ time, this is also the computational effort required by the LV algorithm.

The outline of the rest of this paper is as follows. In Section 3 we develop a dynamic programming algorithm that solves the 2FL problem optimally. We also report the average execution times required by this dynamic programming algorithm. In Section 4 we develop lower bounds and heuristic algorithms for the mFL problem. Also, we perform a computational experiment to compute the average performance of our heuristics on randomly generated problems. In Section 5 we use the heuristics developed in Section 4 to decompose the HFS_{m_1,m_2} design into flowline-like units. Then, we perform an experiment to assess the throughput performance of mFL in comparison with HFS_{m_1,m_2}. We conclude in Section 6 with guidelines on the formation of flexible flowshop manufacturing systems.

3 The $2FL$ problem

A set of jobs $J = \{J_1, J_2, \ldots, J_n\}$ is given and every job J_i consists of two tasks, with processing time requirements a_i and b_i. We will use $a_i,\ b_i$ to denote both the tasks and the requirements of job J_i. All jobs are assumed to be available at time zero and no preemption is allowed for the tasks. Each job in J must be processed exclusively by one of two available flowlines L_1, L_2, where each flowline is a 2-machine flowshop; see Figure 1. To maximize throughput, as well as the machine utilization of the 2FL system, we are interested in scheduling the jobs in J to the two flowlines $L_1,\ L_2$, so that the resulting schedule minimizes makespan. Hence, the 2FL problem is equivalent to partitioning the job set J into two subsets of jobs, say I_1 and I_2, and then dedicate the flowline L_k to the subset $I_k,\ k = 1, 2$. After resolving this assignment problem, scheduling on L_1 and L_2 is a simple task involving Johnson's algorithm for minimizing makespan on a 2-machine flowshop; see Johnson, 1954.

Observe that in case that $b_i = 0$ for all $J_i \in J$, our $2FL$ problem reduces to the problem of minimizing makespan on two identical parallel machines. This scheduling problem is known to be ordinary $\mathcal{NP}$-complete (see Garey and Johnson, 1979), and therefore our problem, since it contains the above problem as a special case, is $\mathcal{NP}$-complete as well.

Assume that the set $J = \{J_1, J_2, \ldots, J_n\}$ is ordered according to Johnson's order. Define the quantities:

$p_i = a_i + b_i.$

$A_i = \sum_{j=1}^{i} a_j.$

$f_i(I, S_1, S_2)$ = the optimal makespan value of 2FL for the jobs $\{J_1, J_2, \ldots, J_i\}$, when the amount of idle time on M_{11} is I, and the idle time after the last job of M_{21} and M_{22} is S_1 and S_2 respectively.

By definition, $S_1 \cdot S_2 = 0$ since the makespan value is attained on at least one of M_{21} and M_{22}. More specifically, if the makespan is attained on M_{21} we have $S_1 = 0$, and if it is attained on M_{22} we have $S_2 = 0$. Also, in the above definition of $f_i(I, S_1, S_2)$, the variables $I,\ S_1,\ S_2$ take values from the interval $[0, P_n]$, where $P_n := \sum_i p_i$. These observations indicate that the state space of the dynamic program (DP) to calculate $f_i(\cdot, \cdot, \cdot)$ is $\mathcal{O}(nP_n^2)$.

The following DP algorithm is based on the fact that the optimal makespan $f_i(I, S_1, S_2)$ is attained on at least one of M_{21} and M_{22} before the scheduling of J_i, and on at least one of M_{21} and M_{22} after the scheduling of J_i ; thus producing 4 possible combinations.

Definition 1 *Let C_{kr} be the combination where the value $f_i(I, S_1, S_2)$ is attained by L_r after the scheduling of J_i, and by L_k prior to scheduling J_i, $k, r \in \{1, 2\}$, $k \neq r$.*

For each C_{kr}, we depict in Figure 2 the alternative schedule configurations for the job set $\{J_1, J_2, \ldots, J_{i-1}\}$, that can result to C_{kr} after the insertion of J_i.

INSERT FIGURE 2 HERE

Evidently, there are two alternative schedule configurations for C_{11}. In C_{11} a), J_i is assigned to L_1, and in C_{11} b), J_i is assigned to L_2 (in Figure 2, the dotted boxes indicate the job J_i). Similarly, there are two configurations for C_{22}. In C_{21}, the makespan is attained on L_2 prior to inserting J_i, and hence J_i must be inserted into L_1 if the makespan is to be attained on L_1 after the insertion. Hence, there is a single configuration for C_{21}. Similarly, there is a unique configuration for C_{12}. The four C_{kr} combinations motivate the following recurrence relation.

Recurrence Relation: Let $J_1, J_2, \ldots, J_n$ be the set of jobs ordered according to Johnson's order. Then,

$$f_i(I, 0, S_2) =$$

$$= \min \begin{cases} C_{11} : \min \begin{cases} \min_{I' \leq a_i} \{f_{i-1}(I', 0, S_2 + I' - p_i) - I'\} + p_i & \text{if } I = b_i \\ f_{i-1}(I - b_i + a_i, 0, S_2 - b_i) + b_i & \text{if } I > b_i \\ \min_{S_2' \geq S_2} \{f_{i-1}(I, 0, S_2') : S_2' = S_2 + b_i + (A_i - 2f_{i-1}(I, 0, S_2') + I + S_2')^+\} & \end{cases} \\ C_{21} : \begin{cases} f_{i-1}(I - S_2 + a_i, b_i - S_2, 0) + S_2 & \text{if } I > b_i \\ \min_{0 \leq S_1' \leq a_i + b_i} f_{i-1}(p_i - S_2, S_1', 0) + S_2 & \text{if } I = b_i \end{cases} \end{cases}$$

$$f_i(I, S_1, 0) =$$

$$= \min \begin{cases} C_{12} : \min \begin{cases} \min_{0 \leq S_2' \leq b_i} f_{i-1}(I - S_1, 0, S_2') + S_1 & \\ \min_{0 \leq S_2' \leq a_i + b_i - S_1} f_{i-1}(I - S_1, 0, S_2') + S_1 & \text{if } A_i + I - S_1 + S_2' > 2f_{i-1}(I - S_1, 0, S_2') \\ 0 & \text{and } (A_i + I - S_1 + S_2' - 2f_{i-1}(I - S_1, 0, S_2'))^+ + b_i = S_1 + S_2' \end{cases} \\ C_{22} : \begin{cases} \min_{S_1' \geq I} f_{i-1}(I + a_i, S_1', 0) & \text{if } I = S_1 + b_i \\ f_{i-1}(I + a_i, S_1 + b_i, 0) & \text{if } I > S_1 + b_i \\ \min_{0 \leq y \leq a_i} \{f_{i-1}(I - b_i - y, S_1 - b_i - y, 0) + y\} + b_i & \text{if } y = (A_i - 2f_{i-1}(I - b_i - y, S_1 - b_i - y, 0) + I')^+ \end{cases} \end{cases}$$

Boundary Conditions:

$$f_1(I, S_1, S_2) = \begin{cases} p_1 & \text{if } (I, S_1, S_2) = (b_1, 0, p_1) \\ p_1 & \text{if } (I, S_1, S_2) = (p_1, p_1, 0) \\ \infty & \text{otherwise} \end{cases}$$

Optimal Solution:

Let $f_n^\star$ denote the optimal solution of DP with the above boundary conditions. Then

$$f_n^\star = \min_{I, S_1, S_2} f_n(I, S_1, S_2).$$

Theorem 1 *The recurrence relation for $f_i(I, S_1, S_2)$, along with the above boundary conditions, produce the optimal makespan value for the $2FL$ problem.*

All proofs are included in the appendix.

Complexity of the DP algorithm:

As indicated earlier, the state space of DP is $\mathcal{O}(nP_n^2)$. It is easy to check that the effort required at every iteration of the DP is of order $\mathcal{O}(P_n)$. Therefore, the complexity of the DP is $\mathcal{O}(nP_n^3)$.

3.1 Computational Performance of The DP Algorithm

We coded the DP algorithm in C++ and tested its computational efficiency on a Pentium 133 processor. For each of the problem sizes $n = 20, 30, 40$ and 50 we randomly generated 50 test problems and computed the amount of time required to solve each problem. In Table 1 we report the average (over the 50 problems) computational time for each value of n. To examine the effect of the variability in processing time durations into computational efficiency, we experimented with the ranges $[1, 10]$ and $[1, 20]$ for the durations of a_i and b_i. For example, using the range $[1, 10]$, we randomly selected a value for a_i from a discrete uniform distribution on $[1, 10]$. Similarly for the range $[1, 20]$.

INSERT TABLE 1 HERE

As evidenced by Table 1 the DP algorithm requires an average of 162.5 seconds for the 8 size/ratio combinations considered. As expected, the range $[1, 20]$ results to greater CPU times due to the dramatic increase in the number of states of the dynamic program. The average over the problems generated for the range $[1, 10]$ is 36.8 seconds, while the corresponding average for the range $[1, 20]$ is 288.2 seconds. We can observe that an increase of n by 10, results to an increase in CPU times by a factor of roughly 3. Since $n = 50$ is considered a relatively large problem size, and given the CPU times of Table 1, the efficiency of the DP algorithm appears to be adequate for most practical applications.

4 Lower Bounds and Heuristics for the mFL Problem

In this section we develop lower bounds for the mFL problem in subsection 4.1. In subsection 4.2 we develop a number of heuristic algorithms that exploit a variety of characteristics of the mFL system. Our heuristics are evaluated against the lower bounds of subsection 4.1, by computing the average relative gaps on randomly generated problems. Our experiment is reported in 4.3.

4.1 Lower bounds

Given the set J of jobs, we can construct an auxilliary 2-machine flowshop problem (AFS) for mFL, by replacing a_i by $\frac{1}{m}a_i$, and b_i by $\frac{1}{m}b_i$. Hence, the AFS problem is a makespan problem on a single flowline. Let C_{AFS} be the optimal makespan value obtained by the application of Johnson's algorithm on AFS (see Johnson, 1954). Then, if C_{mFL} denotes the optimal makespan value for mFL, we have the following result.

Lemma 1 $C_{AFS} \leq C_{mFL}$.

The above lemma is used to develop a better lower bound for mFL, when m is a power of 2; i.e. $m = 2^k$ for $k \geq 1$. As we will show, the case $m = 2^k$ is not restrictive at all, since with minor additional computational effort we can transform problems with $2^{k-1} < m < 2^k$ to equivalent problems with $m = 2^k$.

The case $m = 2^k$

Let AFL be the auxilliary problem on two flowlines (2FL) where a_i is replaced by $2a_i/m$ and b_i is replaced by $2b_i/m$. Let us denote by C_{LB} the makespan value obtained by the application of DP on AFL. Then, we have the following result.

Theorem 2 $C_{LB} \leq C_{mFL}$.

The case $m \neq 2^k$

Consider the case where $2^{k-1} < m < 2^k$. Then, the mFL problem can be transformed into an equivalent $(2^k)FL$ problem by adding to the job set J, $2^k - m$ dummy jobs with processing requirements $(0, B)$, and $2^k - m$ dummy jobs with processing requirements $(B, 0)$, where $B = C_{mFL}$. Then, an optimal schedule S^{2^k} for $(2^k)FL$ induces an optimal schedule for mFL by disregarding the $2^k - m$ flowlines of S^{2^k} that are assigned to process the $2(2^k - m)$ dummy jobs. Since the optimal makespan C_{mFL} is unknown, we can perform bisection search on B, in the range

$$B \in [\frac{1}{m}\sum_{i=1}^{n} p_i, \sum_{i=1}^{n} p_i].$$

In this construction, the optimal makespan of the mFL problem, is the least value of B for which $C_{mFL} = B$.

We can use the above observation to adapt Theorem 2 to the case where $m \neq 2^k$, by applying our DP algorithm to the AFL problem on the revised job set that except for $(2a_i/m, 2b_i/m)$, $1 \leq i \leq n$, includes $2^k - m$ dummy jobs with processing requirements $(0, B')$ and $2^k - m$ dummy jobs with processing requirements $(B', 0)$. In this case we have that

$$B' \in [\frac{2}{m^2}\sum_{i=1}^{n} p_i, \frac{2}{m}\sum_{i=1}^{n} p_i],$$

and the optimal value for B' is the least value for which $C_{LB} = B'$. Since the computational effort required by DP is $\mathcal{O}(nP_n^3)$, the described bisection search scheme can provide a lower bound to the mFL problem for $m \neq 2^k$ in $\mathcal{O}(nP_n^3 \log P_n)$ time.

4.2 Heuristic algorithms

In this subsection we develop and test heuristic algorithms for the mFL problem. Later on we compute the relative deviation of the heuristics from the lower bound C_{LB}. Our first heuristic combines the two tasks a_i and b_i of each job into a single task $p_i = a_i + b_i$, and partitions J into m parts. Each part is then assigned to a single flowline of mFL.

Heuristic H_1

1. Let S be the Johnson's order for the jobs.

2. Apply the first available machine rule to assign the jobs $p_i = a_i + b_i$, $1 \leq i \leq n$, on m parallel identical machines (mP). Let A_k be the jobs allocated to machine M_k of mP, $1 \leq k \leq m$.

3. Schedule the jobs in A_k onto the k-th flowline L_k, $1 \leq k \leq m$, according to Johnson's order.

The computational effort required by steps 2 and 3 of H_1 is $\mathcal{O}(n)$ and hence the complexity of H_1 is $\mathcal{O}(n \log n)$ due to the sorting required at step 1. The following heuristic is a refinement of the FAM rule for the mFL environment. Namely, H_2 assigns the first unscheduled job, say J_i, onto the first available flowline (FAFL rule); i.e. to the flowline that results to the least makespan value after the insertion of J_i.

Heuristic H_2

1. Let S be the Johnson's order for the jobs.

2. Apply the FAFL rule with respect to the order S.

The computational effort required by step 2 of H_2 is $\mathcal{O}(n)$ and hence the complexity of H_2 is $\mathcal{O}(n \log n)$ due to the sorting required at step 1. A different line of heuristic schedules is obtained by our next heuristic where we start off with the Johnson order with respect to the original processing times of (a_i, b_i). Then, we break the resulting schedule in m chunks each of which is assigned to a flowline.

Heuristic H_3

1. Let S be the Johnson's order for the jobs.

2. Apply bisection search on C, in the range $[\frac{1}{m}\sum_i p_i, \sum_i p_i]$.

3. for $k := 1$ to m do
   assign on L_k as many leading unscheduled jobs of S as can fit in the time interval $[0, C]$.

The above heuristic is analogous to the multifit algorithms for parallel identical machine scheduling (a survey of such algorithms is provided in Cheng and Shin, 1990). If a feasible schedule (i.e. a schedule where all jobs are assigned to the m machines) can be found on C periods, then the bisection search will try smaller values. Else, the search will be limited to values greater than C. Note that the schedule produced by H_3 assigns to each flowline a chunk of consecutive jobs of the Johnson's schedule with respect to processing times (a_i, b_i). Step 2 of H_3 results to $\mathcal{O}(\log P_n)$ trial makespan values, and step 3 requires $\mathcal{O}(n)$ computational effort. Since the effort required by step 1 is $\mathcal{O}(n \log n)$, the complexity of H_3 is $\mathcal{O}(n \log P_n)$.

Our next heuristic exploits the allocation produced by the application of the DP on the AFL problem. Since the AFL problem led us to a lower bound for mFL, it seems reasonable to use it for the construction of near optimal schedules. In the following description we assume that m is a power of 2; i.e. $m = 2^k$. If not, we described in Section 4.1 how we can transform our problem to an equivalent one that satisfies the power of 2 condition.

Heuristic H_4

The tree T of Figure 3 facilitates our description of H_4. At level 0 we apply DP with respect to the processing times $(2a_i/m, 2b_i/m)$. The DP algorithm partitions the job set J to two subsets. The jobs that are going to be processed by L_1, and the jobs that are going to be processed by L_2 (recall that the DP solves optimally the 2FL problem). Those jobs allocated to L_1 correspond to the jobs that will be processed on the upper half ($m/2$ machines) of mFL; we refer to this subproblem as F_1^k, where $k = \log_2 m$. The jobs allocated to L_2 by DP are the jobs that will be processed on the lower half of mFL; we refer to this subproblem as F_2^k. At level 1, we apply the DP algorithm on the subproblem F_1^k; thus allocating jobs to the first and second quarter of machines of mFL. Similarly, the application of the DP on F_1^k allocates jobs to the third and fourth quarter of machines of mFL. In every level of the tree T of Figure 3 we indicate the processing times utilized by DP. The leaves of T indicate the allocation of jobs to single flowlines and hence the DP algorithm is utilized only by the nodes of the levels 0 through $k-1$ of T. A schedule is provided for each of the m flowlines represented by leaves, from the DP applications of the previous level. There are $2^k - 1$ or $\mathcal{O}(m)$ nodes in T that are not leaves. Since DP is applied once for each such node and since each application of DP requires $\mathcal{O}(nP_n^3)$ time, the complexity of H_4 is $\mathcal{O}(mnP_n^3)$.

INSERT FIGURE 3 HERE

Heuristic H_5

To reduce the computational requirements while capturing the main idea of H_4, we develop the heuristic H_5. This heuristic is identical to H_4 except that the application of the DP algorithm on each node of T, is replaced by H_1, H_2, and H_3. Subsequently, we use the minimum makespan schedule among the 3 schedules obtained by H_1, H_2, and H_3. Arguing as before, the complexity of H_5 is $\mathcal{O}(mn \log n)$.

In the next subsection we report the average performance of all of the above heuristics.

4.3 Computational performance

To evaluate the performance of our heuristics we performed an experiment where we considered test problems with $m = 2, 4$, or 8 flowlines, and $n = 20, 30, 40$, or 50 jobs. For each combination of m, n values we randomly generated 50 problems using the ranges $[1, 10]$ and $[1, 20]$ for the processing time durations a_i and b_i. These durations were drawn as described in Section 3. For each of the 50 test problems, we recorded the relative gap $\frac{C_H - C_{LB}}{C_{LB}}$. To compute this gap we compute C_H using the heuristic $H \in \{H_1, H_2, H_3, H_4, H_5\}$, and the lower bound C_{LB} described in Theorem 2. We also recorded the best gap (denoted by *Best of* $H_1 - H_3, H_5$ among the 4 heuristics H_1, H_2, H_3 and H_5. The heuristic H_4 was not included in the calculation of Best H, because H_4 has non-polynomial complexity; recall that it utilizes $m - 1$ applications of the DP algorithm along with the tree structure of Figure 3. In Table 2 we report the average (over the 50 problems) relative gaps for all the m, n and range combinations considered.

Note that the case $m = 2$ corresponds to evaluating the algorithms H_1, H_2, H_3, and H_5, as heuristic approaches for the 2FL problem that can be solved optimally using the DP algorithm. Hence, for $m = 2$ the reported relative gaps indicate, in fact, relative deviation from optimality. Observe that H_4 coinsides with the DP algorithm for $m = 2$, and hence the resulting deviations equal 0.

TABLE 2 GOES HERE

The most important finding presented by Table 2 is that H_4 has near optimal performance with an average relative gap of 3.3% for the range $[1, 10]$ and 4.1% for the range $[1, 20]$; these averages are taken over the combinations where $m = 4$ and $m = 8$ only. This means that although H_4 may be time consuming (especially for larger values of n), it provides an excellent solution approach for mFL.

The heuristics H_1 and H_2 have very similar performance. The heuristic H_3 is inferior to H_1 and H_2 for the combinations where $m = 2$ and $m = 4$. In general, none of the heuristics H_1, H_2 and H_3 consistently outperforms the others.

As expected, the heuristic H_5 dominates the heuristics H_1, H_2 and H_3; recall that H_5 uses the tree T of Figure 3 where each node uses the best solution among the ones obtained by H_1, H_2 and H_3. However, since the subproblems corresponding to each node of T consider processing times of the form $(\frac{a_i}{2^k}, \frac{a_i}{2^k})$ rather than the actual processing times of (a_i, b_i), it is not unlikely that one of H_1, H_2 or H_3 performs better than H_5. As a result, the values recorded for $Best\ of\ H_1 - H_3, H_5$ in Table 2, although close to the corresponding values of H_5, are not identical.

Focusing on the results obtained for Best H, we conclude that it has satisfactory performance for all combinations except for $m = 8$ and $n = 20, 30$. All other combinations provide an overall average gap of 11.3%. Evidently, $Best\ of\ H_1 - H_3, H_5$ has unsatisfactory performance for (m, n) combinations with large values of m and small values of n. For such combinations we suggest using H_4 since the corresponding CPU requirements are not high; recall from Table 1 that the time requirements of DP are low for small values of n.

In conclusion, the results obtained for $Best\ of\ H_1 - H_3, H_5$, and H_4 indicate that by using the right algorithms for each (m, n) combination, we can solve fast and accurately mFL problems with relatively large values of m and n.

5 Decomposing Hybrid Flowshops to Flowline-like Designs

The routing flexibility of the HFS_{m_1,m_2} environment often comes in the expense of sophisticated material handling expenditures for automated transfer lines, automated guided vehicles, and the managing technology required for this equipment. In this section we offer an alternative flowshop design for HFS_{m_1,m_2} with simpler routing structure and only minor loss in throughput performance. Namely, we consider the decomposition of HFS_{m_1,m_2} into m_1 smaller independent units (we assume $m_1 \leq m_2$) each of which is a hybrid flowshop of the form $HFS_{1,k}$, where k is an appropriately selected number (see Figure 1). The assumption $m_1 \leq m_2$ is not restrictive since the case $m_1 > m_2$ is symmetric and hence all the results developed in this section apply.

The above decomposition significantly simplifies the managing of HFS_{m_1,m_2}, by decomposing it into smaller manufacturing cells. As a result, the routes available to a job coming off stage 1 are significantly less than in HFS_{m_1,m_2} which usually translates to significant savings in material handling costs. However, the above savings come in the expense of a slight deterioration in throughput performance. In this section we show that in most cases the throughput loss is insignificant compared to the benefits provided by the simplicity of the material handling structure and the associated cost savings.

Intuitively it is beneficial to distribute the workload of HFS_{m_1,m_2} so that each machine can handle about the same workload. This is in line with similar conclusions for scheduling environments where balanced designs result to improved throughput performance. For this reason, we assume that the number k of machines in $HFS_{1,k}$ is either $\lceil \frac{m_2}{m_1} \rceil$ or $\lceil \frac{m_2}{m_1} \rceil - 1$ (although our approach is applicable to any arbitrary decomposition). To further simplify our exposition, we assume that m_2 is an integral multiple of m_1, i.e. $m_2 = km_1$. If a HFS_{m_1,m_2} design does not possess this property, we can introduce $km_1 - m_2$ dummy stage 2 machines along with $km_1 - m_2$ dummy jobs with processing time requirements $(0, B)$, where B is a trial makespan value. As in Section 4.1, bisection search on B can identify the least value of B for which there exists a feasible decomposition of HFS_{m_1,m_2} into m_1 units of the form $HFS_{1,\frac{m_2}{m_1}}$, so that the resulting makespan value equals B. Moreover, without loss of generality we can retain our assumption that m_1 is a power of 2. We refer to the decomposition problem as PHFS since it decomposes the HFS design to parallel HFS sub-designs.

All the algorithms developed in Section 4.2 for the mFL problem can be extended to the PHFS as follows.

Heuristics for PHFS

1. Apply the heuristic H, $H \in \{H_1, H_2, H_3, H_4, H_5\}$ to the m_1FL problem with respect

to the processing times $(a_i, \frac{m_2}{m_1} b_i)$. Let A_r be the set of jobs assigned to L_r, $1 \leq r \leq m_1$.

2. For $r = 1$ to m_1 do
   apply the LV algorithm to schedule the jobs in A_r onto a $HFS_{1,\frac{m_2}{m_1}}$ unit.

At Step 1 of the above algorithm we use any of the heuristics H_1, H_2, H_3, H_4 or H_5 to partition the job set J into m_1 parts $A_1, A_2, \ldots, A_{m_1}$. For each $1 \leq r \leq m_1$, we utilize the algorithm LV of Lee and Vairaktarakis, 1995, (see Section 2) to schedule the jobs in A_r onto a $HFS_{1,\frac{m_2}{m_1}}$ sub-design. The LV algorithm appears to have the best known performance for the $HFS_{1,m}$ problem to minimize makespan, with worst case error bound $1-\frac{1}{m}$ and near optimal performance on randomly generated problems (the average relative gaps are less than 0.5%).

Let C_{PHFS} be the optimal makespan value for the PHFS problem. Then, the relative gaps of the heuristics for PHFS described above are attributed to two sources. Namely, suboptimality of the heuristic $H \in H_1, H_2, H_3, H_4, H_5$ for the m_1FL problem, and suboptimality of the LV heuristic for the $HFS_{1,\frac{m_2}{m_1}}$ problem. According to our comment above, the worst case error bound for the latter source of suboptimality is $1 - \frac{m_1}{m_2}$. In Table 3 we report our computational experiment to assess the throughput performance of our heuristics for PHFS. Table 3 was developed in an identical fashion as Table 2, and it reports the relative gap of the makespan C_{PHFS} of our heuristics for the PHFS problem from the lower bound C_{HFL} computed as follows.

Let HFL be the auxilliary problem on two flowlines (2FL) where a_i is replaced by $2a_i/m_1$ and b_i is replaced by $2b_i/m_2$. Then, a proof similar to that of Theorem 2 verifies the following result.

Theorem 3 $C_{HFL} \leq C_{PHFS}$.

In Table 3 the $m_1 \times m_2$ designs considered are 2×4, 2×8, 4×8 and 4×16. The 4×16 design for instance is decomposed by our algorithms to 4 independent designs of the form 1×4. Similarly, all the remaining designs are decomposed to as many subsystems as the number m_1 of stage 1 machines.

INSERT TABLE 3 HERE

The observations stemming from Table 3 are similar to those of Table 2. The heuristic H_4 has superior performance than the best of H_1, H_2, H_3 and H_5. Recall that the computational requirements of H_4 are significantly greater than any of the rest 4 heuristics. It appears that the *Best of* $H_1 - H_3, H_5$ provides reasonable answers for most cases, except when the problem size is small e.g., $n = 20$ or the number $m_1 + m_2$ of total machines in the system is excessively large e.g., when $m_1 \times m_2 = 4 \times 16$. However, in both cases H_4 provides good average relative gaps. The average relative gap of H_4 over all the combinations considered is 3.55%, of which 4% is atrributed to the range $[1, 20]$ for the processing times of a_i and b_i, while the corresponding average gap for the range $[1, 10]$ is 3.10%.

In the next subsection we compare the throughput performance of the two production systems contrasted in this research. Namely, we examine how faster is the HFS_{m_1,m_2} environment, as compared to the corresponding PHFS system that consists of m_1 independent modules of the form $HFS_{1,\frac{m_2}{m_1}}$.

5.1 The effect of routing flexibility on Throughput

It has become clear that obtaining optimal algorithms for the PHFS and HFS_{m_1,m_2} environments is extremely difficult. As a result, obtaining the exact gap of the throughput performance of these two environments is impossible. Nevertheless, we can estimate those gaps by using the H_4 heuristic to obtain C_{PHFS}, and the LV heuristic (see Section 2) to obtain C_{LV}. As documented in Lee and Vairaktarakis, 1995 the LV heuristic provides near optimal solutions for the problem of minimizing makespan in HFS_{m_1,m_2}. Also, this research shows that H_4 provides near optimal solutions for the PHFS problem. Hence, instead of using optimal algorithms (that are unavailable), we use the best available heuristics.

In Table 4 we report the average (over 50 randomly generated problems) relative gaps $(C_{PHFS} - C_{LV})/C_{LV}$ for the problem sizes $n = 20, 30, 40$ and 50 jobs, and the designs 2×2, 2×4, 2×8, 4×4, 4×8, and 4×16. The 2×4 design for instance, is viewed by $HFS_{2,4}$ as a two stage system with free routing between the 2 stage 1 machines and the 4 stage 2 machines, while the corresponding PHFS system is viewed as 2 independent $HFS_{1,2}$ systems. Similarly for the remaining $m_1 \times m_2$ values. In this experiment, for every randomly generated problem the C_{PHFS} and C_{LV} makespan values are obtained for the same scenario of processing times. These processing times are uniformly selected from the range $[1, 20]$.

INSERT TABLE 4 HERE

It becomes apparent that the throughput performance gaps of PHFS and HFS_{m_1,m_2} are surprisingly small, and they become negligible as the number of jobs increases. In particular, for $n = 50$ the average throughput performance gap is 1.33%. In light of the fact that the corresponding gap for $n = 20$ is 4.06%, it may be marginally beneficial to use the HFS_{m_1,m_2} system for batches of 20 or fewer jobs. Another trend exhibited by the data of Table 4 is that the performance gaps increase with the total number of machines in the system, $m_1 + m_2$.

Overall, our results tend to indicate that investing in sophisticated material control structures and multidirectional routing offers minute throughput increases over a unidirectional routing structure that appropriately decomposes the machines to independent production cells.

6 Conclusion

This paper examined the ramifications of free routing between adjacent stages of flowshop production systems. We found that the throughput benefits of such designs are marginally better than flowline-like designs that employ unidirectional routing and decomposition of the production system into small independent cells. We provided algorithms that can be readily utilized to attain flowline-like decompositions within the limits of today's computing capabilities. These algorithms are supported by detailed computational experiments. Our future research will be directed towards investigating the value of other forms of flexibility (such as material handling flexibility) for flowshop as well as other manufacturing protocols.

References

[1] Afentakis, P. (1986). Maximum throughput in Flexible Manufacturing Systems. *Proceedings of the Second ORSA/TIMS Conference on Flexible Manufacturing Systems: Operations Research Models and Applications*, Elsevier Science Publishers B.V., Amsterdam, 509-521.

[2] Baker, K.R. (1993). Elements of Sequencing and Scheduling, *Dartmouth Hanover*, NH 03755.

[3] Blakewicz, J., K. Echer, G. Schmidt and J. Weglartz (1993). Scheduling in Computer and Manufacturing Systems. *Springer Verlag*, Berlin, Germany.

[4] Cheng, T.C.E. and C.C.S. Sin (1990). A State-Of-The-Art Review Of Parallel-Machine Scheduling Research. *European Journal of Operational Research*, 47:271-192.

[5] Erschler, J, F. Roubellat and C. Thuriot (1985). Steady State Scheduling of a Flexible Manufacturing System with Periodic Releasing and Flow Time Constraints. *Annals of Operations Research* 3:333-353.

[6] Garey, M.R. and D.S. Johnson (1979). Computers and Intractability. *W.H. Freeman*, San Francisco, CA.

[7] Ghosh, S. and C. Gaimon (1992). Routing Flexibility and Production Scheduling in a Flexible Manufacturing environment. *European Journal of Operational Research*, 60:344-364.

[8] Guinet, A. and M.M. Solomon (1996). Scheduling Hybrid Flowshops to Minimize Maximum Tardiness or Maximum Completion Time. *International Journal of Production Research* 34:1643-1654.

[9] Gupta, J.N.D., A.M.A. Hariri and C.N. Potts (1997). Scheduling a Two-Stage Hybrid Flowshop with Parallel Machines at the First Stage. *Annals of Operations Research*, series on *Mathematics of Industrial Systems* 69:171-191.

[10] He, D.W., A. Kusiak and A. Artiba (1996). A Scheduling Problem in Glass Manufacturing. *IIE Transactions* 28:129-139.

[11] Hoogeveen, J.A., J.K. Lenstra and B. Veltman (1996). Preemptive Scheduling in a Two-Stage Multiprocessor Flowshop is NP-hard. *European Journal of Operational Research*, 89:172-175.

[12] Johnson, S.M. (1954). Optimal Two and Three-Stage Production Schedules with Setup Times Included. *Naval Research Logistics Quarterly* 1:61-68.

[13] Kouvelis P. and G. Vairaktarakis (1998). Flowshops with Processing Flexibility Across Production Stages, *IIE Transactions*, 30(8):735-746.

[14] Lee, C.-Y. and G. Vairaktarakis (1994). Minimizing Makespan in Hybrid Flowshops. *Operations Research Letters*, 16:149-158.

[15] Lee C.-Y. and G. Vairaktarakis (1998). Performance Comparison of Some Classes of Flexible Flowshops and Job Shops, *International Journal of Flexible Manufacturing Systems*, 10(4):379-405.

[16] Maleki, R.A. (1991). Flexible Manufacturing Systems: The Technology and Management. *Prentice Hall*, Englewood Cliffs, NJ.

[17] Pinedo, M. (1995). Scheduling: Theory, Algorithms and Systems. *Prentice Hall*, Englewood Cliffs, NJ.

[18] Shanker K. and Y.T. Tzen (1985). A Loading and Dispatching Problem in a Random Flexible Manufacturing System. *International Journal of Production Research* 23(3):579-595.

[19] Solomon, M.M., P.K. Kedia and A. Dussauchoy (1996). A Computational Study of Heuristics for Two-Stage Flexible Flowshops. *International Journal of Production Research* 34:1399-1416.

[20] Stecke, K.E. (1985). Design, Planning, Scheduling and Control Problems of Flexible Manufacturing Systems. *Annals of Operations Research* 3:3-12.

[21] Stecke, K.E. (1992). Planning and Scheduling Approaches to Operate a Particular FMS. *European Journal of Operational Research* 61:273-291.

[22] Wittrock, R.J. (1988). An Adaptable Scheduling Algorithm for Flexible Flow Lines. *Operations Research* 36:445-453.

Appendix

Proof of Theorem 1:

Throughout this proof we refer to Figure 2. We denote by $f_{i-1}(I', S_1', S_2')$ the makespan value of a schedule prior to the insertion of J_i, and by $f_i(I, S_1, S_2)$ the corresponding makespan after the insertion of J_i. The cases C_{11}, C_{12}, C_{21}, and C_{22} are analyzed separately below.

Case C_{11}

In this case we distinguish the subcases where a) J_i is scheduled on L_1, and b) J_i is scheduled on L_2.

subcase a): In this subcase we have that $I \geq b_i$. If $I = b_i$, then we must have $a_i \geq I'$. Then, the idle time on M_{22} increases by $b_i + a_i - I'$ after the insertion of J_i, and hence $S_2 = S_2' + p_i - I'$. Also, the makespan of M_{21} increases by $p_i - I'$. Hence,

$$f(I, 0, S_2) = \min_{I' \leq a_i} f_{i-1}(I', 0, S_2 - p_i + I') + p_i - I' \quad if \quad I = b_i.$$

On the other hand, if $I > b_i$, then we must have that $a_i < I'$. More specifically, in this case we have that $I' = I - b_i + a_i$. Also, the makespan of M_{22} increases by b_i and hence $S_2' = S_2 - b_i$. Therefore,

$$f(I, 0, S_2) = f_{i-1}(I - b_i + a_i, 0, S_2 - b_i) + b_i \quad if \quad I > b_i.$$

subcase b): When J_i is scheduled on L_2, we have that $I' = I$ and

$$f_i(I, 0, S_2) = \min_{S_2' \geq S_2} f_{i-1}(I, 0, S_2'),$$

where

$$S_2' = S_2 + b_i + (A_i - 2f_{i-1}(I, 0, S_2') + I + S_2')^+.$$

The term $(A_i - 2f_{i-1}(I, 0, S_2') + I + S_2')^+$ indicates the idle time induced to M_{22} by the scheduling of a_i on M_{12}.

Combining the 2 subcases of C_{11} we get the recurrence relation:

$$f(I, 0, S_2) = \min \begin{cases} \min_{I' \leq a_i} \{f_{i-1}(I', 0, S_2 + I' - p_i) - I'\} + p_i & \text{if } I = b_i \\ f_{i-1}(I - b_i + a_i, 0, S_2 - b_i) + b_i & \text{if } I > b_i \\ \min_{S_2' \geq S_2} \{f_{i-1}(I, 0, S_2') : S_2' = S_2 + b_i + (A_i - 2f_{i-1}(I, 0, S_2') + I + S_2')^+\} & \end{cases}$$

Case C_{21}

This case holds only iff

$$\max\{f_{i-1}(I', S_1', 0) - I' + a_i, f_{i-1}(I', S_1', 0) - S_1'\} + b_i \geq f_{i-1}(I', S_1', 0) \tag{1}$$

or equivalently,

$$\max\{p_i - I', b_i - S_1'\} \geq 0.$$

We distinguish two subcases for C_{21}; namely i) $I > b_i$ and ii)$I = b_i$. In i), we have that $b_i = S_1' + S_2$, and $I + a_i = I' + S_2$. Hence,

$$f_i(I, 0, S_2) = f_{i-1}(I + a_i - S_2, b_i - S_2, 0) + S_2 \quad if \quad I > b_i.$$

In subcase i), the relation (1) is equivalent to $\max\{S_2 - I + b_i, S_2\} \geq 0$ which holds true because $S_2 \geq 0$ by definition.

In subcase ii) the idle time inserted on M_{21} after the insertion of J_i is $a_i - (I' - S_1')$, and hence $S_1' + S_2 = b_i + a_i - (I' - S_1')$ or $I' = p_i - S_2$. Also, it is true that $S_1' \leq a_i + b_i$ and hence

$$f_i(b_i, 0, S_2) = \min_{0 \leq S_1' \leq a_i + b_i} f_{i-1}(p_i - S_2, S_1', 0) + S_2 \quad if \quad I = b_i.$$

In this subcase, the relation (1) is equivalent to $\max\{S_2, b_i - S_1'\} \geq 0$ which holds always true.

Combining the recurrence relations for the subcases i) and ii), we get

$$f_i(I, 0, S_2) \begin{cases} f_{i-1}(I - S_2 + a_i, b_i - S_2, 0) + S_2 & \text{if } I > b_i \\ \min_{0 \leq S_1' \leq a_i + b_i} f_{i-1}(p_i - S_2, S_1', 0) + S_2 & \text{if } I = b_i. \end{cases}$$

Case C_{12}

This case holds only iff

$$\max\{A_{i-1} - (f_{i-1}(I', 0, S_2') - I') + a_i, f_{i-1}(I', 0, S_2') - S_2'\} + b_i \geq f_{i-1}(I', 0, S_2') \tag{2}$$

Hence, we distinguish the subcases where

i) $A_i - f_{i-1}(I', 0, S_2') + I' \leq f_{i-1}(I', 0, S_2') - S_2'$, and

ii) $A_i - f_{i-1}(I', 0, S_2') + I' > f_{i-1}(I', 0, S_2') - S_2'$.

In i), the insertion of a_i into L_2 does not increase the idle time on M_{22}. In this subcase the relation (2) is equivalent to $b_i \geq S_2'$. We have that $b_i = S_1 + S_2'$, and hence the condition (2) is verified. Also, we have that $I = I' + S_1$. Hence,

$$f_i(I, S_1, 0) = \min_{0 \leq S_2' \leq b_i} f_{i-1}(I - S_1, 0, S_2') + S_1.$$

In ii), the insertion of a_i into L_2 increases the idle time on M_{22}. Since $I = I' + S_1$, the additional idle time on M_{22} is given by $(A_i + I - S_1 + S_2' - 2f_{i-1}(I - S_1, 0, S_2'))^+$. To ensure that only the right combination of values of S_1, S_2' and I are considered by the recurrence relation, we need to ensure that

$$(A_i + I - S_1 + S_2' - 2f_{i-1}(I - S_1, 0, S_2'))^+ + b_i = S_1 + S_2'$$

which indicates that the newly inserted idle time on M_{22} plus b_i equal $S_1 + S_2'$; see Figure 2. With the above observations we get that

$$f_i(I, S_1, 0) = \min_{0 \leq S_2' \leq a_i + b_i - S_1} f_{i-1}(I - S_1, 0, S_2') + S_1,$$

given that $(A_i + I - S_1 + S_2' - 2f_{i-1}(I - S_1, 0, S_2'))^+ + b_i = S_1 + S_2'$ and $A_i + I - S_1 + S_2' > 2f_{i-1}(I - S_1, 0, S_2')$.

The following comment should be made for the range of values of S_2'. Note that $S_1 + S_2' \leq a_i + b_i$ and hence $S_2' \leq a_i + b_i - S_1$. Combining i) and ii) we get the desired recurrence relation for C_{12}.

Case C_{22}

In this case we distinguish the subcases where a) J_i is scheduled on L_1, and b) J_i is scheduled on L_2.

subcase a): This subcase holds iff after the insertion of J_i, M_{22} finishes after M_{12}, i.e.

$$\max\{f_{i-1}(I', S_1', 0) + a_i - I', f_{i-1}(I', S_1', 0) - S_1'\} + b_i + S_1 = f_{i-1}(I', S_1', 0),$$

or equivalently

$$\max\{a_i + b_i - I', b_i - S_1'\} = -S_1. \qquad (3)$$

In this subcase we have that $I' = I + a_i$, and hence the flowshop condition (3) becomes

$$\min\{I, S_1'\} = S_1 + b_i. \qquad (4)$$

Hence,

$$S_1 + b_i = \begin{cases} I & \text{if } I \leq S_1' \\ S_1' & \text{if } I > S_1'. \end{cases}$$

Observing that $f_i(I, S_1, 0) = f_{i-1}(I', S_1', 0)$ in this case, we get the recurrence relation

$$f_i(I, S_1, 0) = \min \begin{cases} \min_{S_1' \geq I} f_{i-1}(I + a_i, S_1', 0) & \text{if } I = S_1 + b_i \\ f_{i-1}(I + a_i, S_1 + b_i, 0) & \text{if } I > S_1' = S_1 + b_i \end{cases}.$$

subcase b): The insertion of J_i into L_2, induces $(A_i - 2f_{i-1}(I', S_1', 0) + I')^+$ units of idle time on M_{22}, and the makespan of M_{22} increases by $b_i + (A_i - 2f_{i-1}(I', S_1', 0) + I')^+$. Define

$$y = (A_i - 2f_{i-1}(I', S_1', 0) + I')^+.$$

Then, we have that $I = I' + y + b_i$, and $S_1 = S_1' + y + b_i$. Also, observe that the idle time y cannot exceed a_i. Hence,

$$f_i(I, S_1, 0) = \min_{0 \leq y \leq a_i} \{f_{i-1}(I - b_i - y, S_1 - b_i - y, 0) + y\} + b_i.$$

Combining the recurrence relations for the above 4 cases, we get the recurrence relation stated in the theorem. This completes the proof of the theorem. □

Proof of Lemma 1:

Let $S^\star$ be an optimal schedule for mFL. Let S_1 be the nondecreasing order of completion times of a_i tasks in $S^\star$. We can use the order S_1 to generate a schedule S for AFS as follows.

Schedule the a_i/m tasks on the upstream machine of AFS according to the order S_1. Then, schedule the b_i/m tasks on the downstream machine of AFS according to the order S_1. The schedule S constructed in this way is not necessarily optimal for AFS and in general it is not a permutation schedule. Let C_S denote the makespan of the schedule S constructed above. Without loss of generality, we reorder the jobs so that $S_1 = \{a_1, a_2, \ldots, a_n\}$. Then, due to the order S_1 we have that the task a_i/m completes in S at time

$$C_i' = \frac{1}{m} \sum_{j \leq i} a_j \leq C_i$$

where C_i is the completion time of a_i on $S^\star$.

For the b_i/m tasks we can assume that they are scheduled contigously so that the last task finishes at time C_{mFL}. Then, the task b_i/m starts in S at time

$$s_i' = C_{mFL} - \frac{1}{m}\sum_{j\geq i} b_j,$$

while b_i starts in $S^\star$ at time $s_i \leq s_i'$, due to the order S_1 and the fact that all b_j tasks with $j \geq i$ start in $S^\star$ after a_i.

Therefore, for every $1 \leq i \leq n$ we have that $C_i' \leq C_i$, and $s_i \leq s_i'$. These relations mean that the flowshop constraints for S are satisfied when the last b_i/m task of AFS is scheduled to finish at time C_{mFL}. Hence, $C_S \leq C_{mFL}$. However, the makespan value C_{AFS} produced by Johnson's algorithm is optimal for the AFS problem and hence $C_{AFS} \leq C_S \leq C_{mFL}$. This completes the proof of the Lemma. □

Proof of Theorem 2:

Let $S^\star$ be an optimal schedule for mFL. Apply Lemma 1 on the first $m/2$ flowlines of the mFL environment. Then, the auxilliary flowshop problem, say AFS_1, is the 2-machine flowshop where the processing times of all tasks assigned to the first $m/2$ flowlines of mFL are multiplied by 2. Equivalently, (a_i, b_i) is replaced in AFS_1 by $(2a_i/m, 2b_i/m)$ iff J_i is assigned to one of the first $m/2$ flowlines of mFL in $S^\star$. Let C_{AFS_1} be the makespan obtained by the Johnson's algorithm for the AFS_1 problem. Then, by Lemma 1 we have that $C_{AFS_1} \leq C_{mFL}$ since C_{mFL} is the makespan of $S^\star$. Similarly, we define the auxilliary flowshop problem AFS_2 for the last $m/2$ flowlines of mFL. Let C_{AFS_2} be the resulting makespan value. Then, $C_{AFS_2} \leq C_{mFL}$.

Note that the schedules obtained by the AFS_1 and AFS_2 problems, provide a feasible solution for the 2FL problem with processing time requirements of $(2a_i/m, 2b_i/m)$, $1 \leq i \leq n$. However, this solution is not necessarily optimal for this 2FL problem. By Theorem 1, the optimal makespan value for the 2FL auxilliary problem equals C_{LB}, and hence $C_{LB} \leq \max\{C_{AFS_1}, C_{AFS_2}\} \leq C_{mFL}$. This completes the proof of the theorem. □

LIST OF TABLES

Size/Range	$[1, 10]$	$[1, 20]$
20	4.5	27.9
30	15.1	92.4
40	41.3	295.4
50	86.3	737.3

Table 1: Average execution times (in seconds) of the DP algorithm for $2FL$.

# of	# of	Heuristics $\frac{C_H - C_{LB}}{C_{LB}}$											
f-lines	jobs	H_1		H_2		H_3		H_5		Best of H_1-H_3, H_5		H_4	
m	n	$[1, 10]$	$[1, 20]$	$[1, 10]$	$[1, 20]$	$[1, 10]$	$[1, 20]$	$[1, 10]$	$[1, 20]$	$[1, 10]$	$[1, 20]$	$[1, 10]$	$[1, 20]$
2	20	3.4	4.3	5.3	4.0	21.7	21.4	2.8	2.9	2.8	2.9	0	0
	30	1.5	3.7	1.2	2.5	20.1	23.8	0.6	1.8	0.6	1.8	0	0
	40	2.4	2.3	1.0	1.4	21.7	21.2	0.7	1.0	0.7	1.0	0	0
	50	6.7	1.6	6.2	1.9	30.8	24.8	2.0	0.9	2.0	0.9	0	0
4	20	17.3	13.2	14.9	15.2	37.6	34.4	13.2	13.8	9.8	11.4	3.3	4.4
	30	11.4	9.5	12.1	9.0	27.3	28.6	9.1	8.8	7.2	7.1	2.8	3.7
	40	9.4	6.2	6.7	5.2	27.2	25.5	6.5	5.7	5.6	3.5	2.1	3.2
	50	4.9	4.9	5.2	3.5	25.9	30.3	3.9	3.9	3.4	2.4	1.9	2.6
8	20	49.3	48.7	48.4	54.1	61.4	56.5	46.6	56.7	35.2	41.4	5.1	6.2
	30	28.8	30.1	34.2	33.7	48.2	46.0	25.7	24.9	21.3	18.6	4.9	5.8
	40	16.9	20.0	15.1	20.4	35.8	41.3	24.8	22.2	12.8	15.9	4.1	4.0
	50	16.7	15.6	15.5	16.1	38.0	40.5	14.5	11.1	11.2	8.4	2.2	3.1

Table 2: Relative gaps of heuristic algorithms for the mFL problem

PHFS design $m_1 \times m_2$	# of jobs n	Heuristics $\frac{C_{PHFS}-C_{HLB}}{C_{HLB}}$ H_1		H_2		H_3		H_5		Best of H_1-H_3, H_5		H_4	
		$[1, 10]$	$[1, 20]$	$[1, 10]$	$[1, 20]$	$[1, 10]$	$[1, 20]$	$[1, 10]$	$[1, 20]$	$[1, 10]$	$[1, 20]$	$[1, 10]$	$[1, 20]$
2×4	20	9.1	10.4	11.2	10.9	26.5	25.8	7.3	6.9	7.4	5.8	0.5	1.8
	30	8.0	6.3	7.1	6.4	23.2	24.7	5.4	6.3	4.5	4.1	0.2	0.8
	40	5.2	4.1	4.2	3.5	22.9	23.2	3.8	4.2	2.9	3.1	0.1	0.5
	50	5.0	3.8	3.9	2.3	28.4	27.4	3.1	1.5	2.4	1.4	0.1	0.3
2×8	20	18.5	15.1	17.2	17.0	47.9	45.9	14.1	14.8	10.7	12.6	3.9	5.1
	30	12.0	11.2	12.9	10.3	36.4	39.2	9.3	9.9	7.4	8.8	3.1	4.4
	40	10.5	7.3	6.9	5.6	31.3	36.1	6.9	6.5	5.7	3.9	2.5	3.6
	50	7.6	5.3	5.8	4.3	32.4	40.4	4.8	4.3	4.2	2.7	2.0	2.8
4×8	20	20.3	17.3	19.4	18.9	59.2	56.3	15.3	16.6	14.4	16.0	5.9	7.4
	30	12.6	12.9	13.6	11.7	45.6	51.2	9.5	10.8	9.1	10.1	4.2	5.5
	40	11.8	8.4	7.3	6.1	40.1	47.4	7.2	7.0	5.9	6.2	3.3	4.0
	50	9.7	6.1	8.4	5.2	41.1	52.2	5.7	4.1	5.2	3.0	1.9	2.5
4×16	20	52.9	51.3	53.4	57.5	68.3	70.4	47.4	57.9	39.2	49.4	7.9	8.7
	30	30.1	32.8	35.1	36.1	59.6	66.3	24.2	23.8	19.6	21.4	6.3	7.7
	40	18.8	22.1	15.8	20.9	46.2	60.4	20.5	22.6	14.8	18.5	4.2	5.0
	50	21.4	16.7	15.6	17.7	45.1	55.2	15.1	12.0	12.1	11.6	3.6	3.9

Table 3: Relative gaps of heuristic algorithms for the PHFS problem

	$\frac{C_{PHFS}-C_{LV}}{C_{LV}}$ Flowline design: $m_1 \times m_2$					
n	2×2	2×4	2×8	4×4	4×8	4×16
20	2.8	1.2	5.2	3.8	5.1	6.3
30	2.1	0.7	3.5	2.9	3.6	4.7
40	1.7	0.3	2.0	2.2	1.8	3.0
50	1.3	0.1	1.4	1.8	1.2	2.2

Table 4: Relative deviation of the throughput performance of Best from HFS_{m_1,m_2}

LIST OF FIGURES

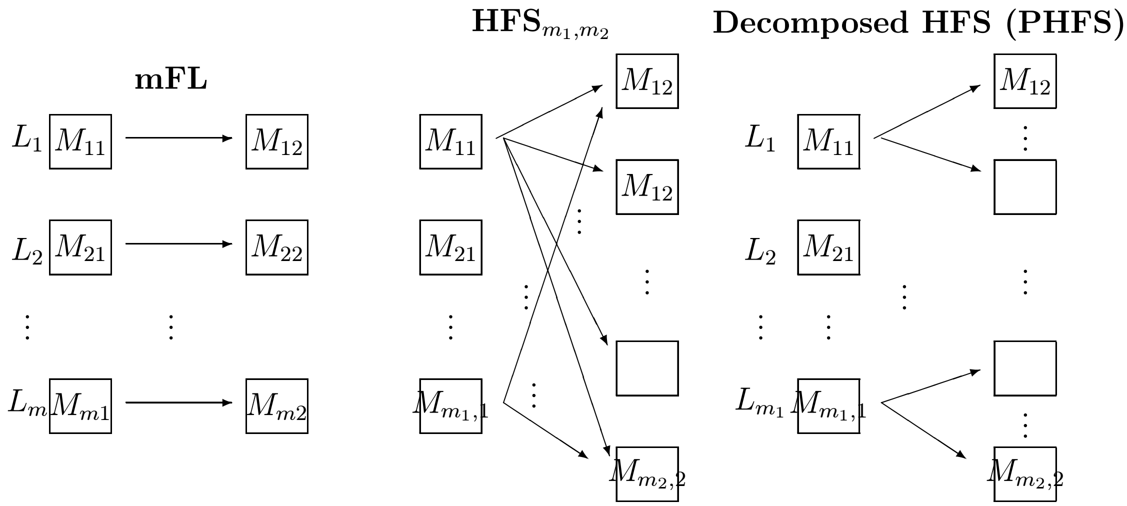

Figure 1: Flowline and hybrid flowshop designs

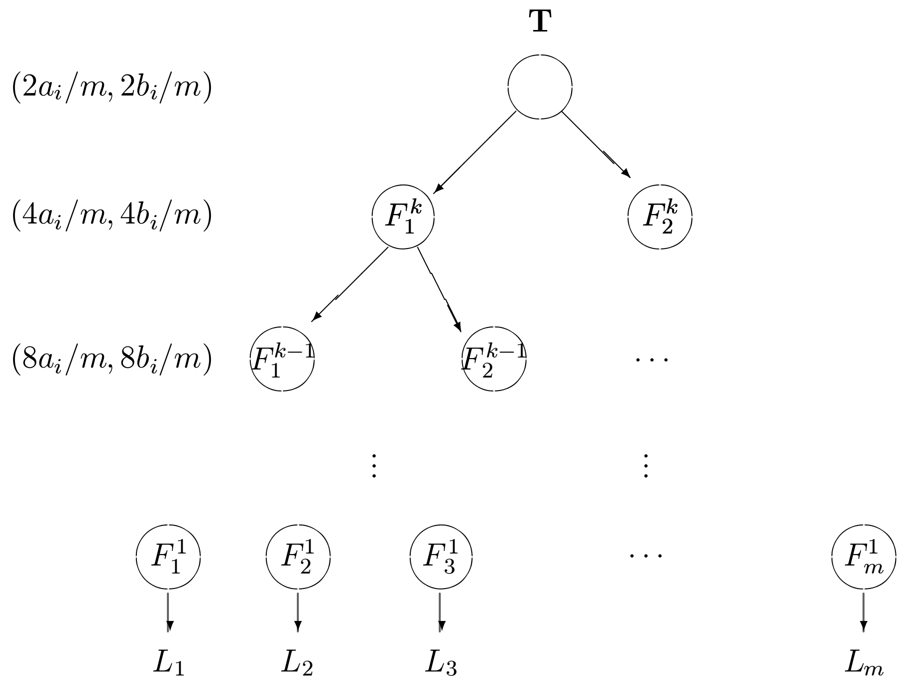

Figure 3: The tree structure of heuristic H_3