ORDER SCHEDULING MODELS: AN OVERVIEW

Joseph Y-T. Leung and Haibing Li
Department of Computer Science
New Jersey Institute of Technology
Newark, NJ 07102, USA
{leung, hl27}@njit.edu

Michael Pinedo
Stern School of Business
New York University
New York, NY 10012, USA
mpinedo@stern.nyu.edu

Abstract Order scheduling models can be described as follows: A machine environment with a number of non-identical machines in parallel can produce a fixed variety of different products. Any one machine can process a given set of the different product types. If it can process only one type of product it is referred to as a dedicated machine, otherwise it is referred to as a flexible machine. A flexible machine may be subject to a setup when it switches from one product type to another product type. Each product type has certain specific processing requirements on the various machines. There are n customers, each one sending in one order. An order requests specific quantities of the various different products and has a release date as well as a due date (committed shipping date). After the processing of all the different products for an order has been completed, the order can be shipped to the customer. This paper is organised as follows. We first introduce a notation for this class of models. We then focus on various different conditions on the machine environment as well as on several objective functions, including the total weighted completion time, the maximum lateness, the number of orders shipped late, and so on. We present polynomial time algorithms for the easier problems, complexity proofs for NP-hard problems and worst case performance analyses as well as empirical analyses of heuristics.

Keywords: order scheduling, models, complexity results, heuristics.

1. INTRODUCTION

We consider a facility with m different machines in parallel. There are k different product types that can be produced on these m machines. Each product type $l = 1, 2, \ldots, k$ can be produced on a subset of the machines, namely $M_l \subseteq \{1, 2, \ldots, m\}$. To produce one unit of type l on a machine $i \in M_l$ requires a processing time t_{li}. When a machine $i \in \{1, 2, \ldots, m\}$ produces a batch of type l products, a setup time of s_{li} is required before the batch can be started. Assume there are n orders from n different clients. Order j requests a quantity $q_{lj} \geq 0$ of product type l, $j = 1, \ldots, n, \quad l = 1, \ldots, k$. For order j, the processing time required to produce items of type l on machine $i \in M_l$ is $p_{lij} = q_{lj} \cdot t_{li}$. Order j may have a release date r_j, a due date (delivery date) d_j, and a positive weight w_j. The completion time of order j, denoted by C_j, is the time when the last product for order j has been completed on one of the machines. Let C_{lj} denote the completion time of the type l production for order j on one of the machines. Clearly,

$$C_j = \max_l \{C_{lj}\}$$

The idea of measuring the overall completion time of an entire set of jobs (i.e. all the jobs required by one order) rather than the individual completion times of each part of any given order is somewhat new. There are several reasons for considering the orders rather than the individual jobs within the orders. First of all, shipping partial orders inevitably causes additional shipping cost. Secondly, it also causes extra management effort. Finally, some customers may require suppliers to ship complete orders. Therefore, suppliers have to wait until all products for an order are ready.

With regard to the completion times $C_1, \ldots, C_n$ of the n orders several objectives are of interest, namely, the makespan $C_{\max}$, the maximum lateness $L_{\max}$, the total weighted completion time $\sum w_j C_j$ of orders, the total weighted tardiness $\sum w_j T_j$ of orders, and the total weighted number of late orders $\sum w_j U_j$.

Clearly, the class of models described above is very rich. Several special cases are of interest, namely:

(i) *The fully dedicated case*: there are m machines and m product types; each machine can produce one and only one type.

(ii) *The fully flexible case*: the m machines are identical and each machine is capable of producing all k products.

(iii) *The arbitrary case*: there are no restrictions on the subsets M_l.

The classes of models have a large number of practical applications. Any Make-To-Order environment at a production facility with a number of flexible resources in parallel gives rise to models of the type described above.

Julien and Magazine (1990) presented two interesting applications of the models described above. The first example involves a manufacturer of computer peripherals such as terminals, keyboards and disk drives. Small businesses that purchase new computer systems order different quantities of each of these peripherals, and often require their entire order to be shipped together. From the manufacturer's point of view, it is advantageous to aggregate the demand of each peripheral and produce large batches in order to minimise the number of setups.

A second example that illustrates the models described is a pharmaceutical company that can produce different types of pills. Each type of pill needs to be bottled separately. However, for a given pill type, it is often necessary to have different bottle sizes. The pills and the bottles may be produced based on forecasts. However, the bottling and packaging stage is order driven. The customers, which may be drugstores or hospitals, order certain quantities of each product type (a product type being a bottle of a given size of a given pill type). The production setups are the switch-overs in the bottling facility.

Yang (1998) presented another example in the form of a car repair shop. Suppose each car has several broken parts that need to be fixed. Each broken part can only be fixed by a certain set of mechanics in the shop. Several mechanics can work on different parts of the same car at the same time. The car will leave the shop when every broken part is fixed.

In manufacturing systems that consist of two stages, different types of components (or subassemblies) are produced first in a pre-assembly stage, and then put together into final products (jobs) in an assembly stage. The pre-assembly stage consists of parallel machines (called feeding machines), each of which produces its own subset of components. Each assembly operation cannot start its processing until all the necessary components are fed in. As shown in several papers (Duenyas, 1994; Lee *et al.*, 1993; Potts *et al.*, 1995), there are many examples of such two-stage assembly systems. An interesting example arises in parallel computing systems, in which several programs (or tasks) are first processed independently on certain processors, and then gathered at a main processor for final data-processing. The main processor can only start its processing after all the programs have fed in their results. As noted by Sung and Yoon (1998), our models only deal with the pre-assembly stage in such two-stage systems.

The general problem has not received much attention in the literature. Some of the concepts underlying the general problem were introduced by Julien and Magazine (1990), who were probably the first to identify this type of problem as order scheduling. The problem was studied in more detail in a dissertation by Yang (1998).

Various special cases of the general problem have been considered in the literature. However, most of these special cases are significantly less compli-

cated than the general problem. An important special case is the case in which each order requests just a single product type. If, in addition, the machines are fully flexible and no product type requires any setup, then the problem immediately reduces to the standard parallel machine environment which is usually referred to as $Pm \mid \beta \mid \gamma$ in the literature. If the machines are not fully flexible and there is no setup time for any product type, the problem reduces to a more general parallel machine environment that is often referred to as *unrelated* machines in parallel ($Rm \mid \beta \mid \gamma$). There is a very rich literature on these parallel machine scheduling problems. For an overview, see Brucker (1995) and Pinedo (2002).

We propose the following notation for our class of scheduling problems. Our notation is an extension of the $\alpha \mid \beta \mid \gamma$ notation introduced by Graham *et al.* (1979). In what follows, we always assume that there are m machines in parallel and n orders that come in from n different customers. The fully dedicated case of this parallel machine environment is denoted by PDm, the fully flexible case by PFm, and the arbitrary case by PAm; when the m is omitted we assume that the number of machines is arbitrary. In the β field we include πk to refer to the fact that we have k different product types; the absence of the k indicates that the number of different product types may be arbitrary. In addition, we include in the β field an s when the setup times for all product types are identical, an s_l when the setup times for the various product types are different but identical for each machine, and an s_{li} when the setup times are dependent on both product types and machines. The absence of s_{li}, s_l, and s indicates that there is no setup time for any product type. Note that setup times do not make sense for the fully dedicated case. In addition, if all machines are identical, then $s_{li} = s_l$ for each machine $i = 1, 2, \ldots, m$. As an example of the notation, $PF6 \mid prmt, s, \pi 3 \mid L_{max}$ refers to the fully flexible case with six machines in parallel and three different product types. Each product type has the same setup time s, order j has a due date d_j, and preemptions are allowed. The objective is the minimisation of maximum lateness.

In the next section, we consider the fully dedicated case. The fully flexible case is considered in Section 3. Finally, we draw some conclusions in the last section.

2. THE FULLY DEDICATED CASE

As mentioned before, in the fully dedicated case, there are m machines and the number of product types is equal to m; each machine can produce only one type. Since machine i is the only machine that can produce type i and type i is the only type that can be produced on machine i, the subscript i refers to a machine as well as to a product type. We note that Wagneur and Sriskandarajah (1993) referred to this model as "open shops with job overlaps", while Ng *et*

al. (2003) called this model "concurrent open shops". This case is considerably easier than the other cases because there is no freedom in the assignment of jobs to machines. Each machine can start processing at time 0 and keeps on producing as long as there is a demand. The main issue here is the assignment of finished products to customers. For dedicated machines, the setup times do not play a role in scheduling, and can therefore be dropped from consideration.

For unweighted objectives, the following structural properties can be shown easily.

Lemma 1 *(i) The makespan $C_{\max}$ is independent of the schedule, provided that the machines are always kept busy whenever there are orders available for processing (i.e. provided unforced idleness is not allowed).*

(ii) If all $r_j = 0$ and $f_j(C_j)$ is increasing in C_j for all j, then there exists an optimal schedule for the objective function $f_{\max}$ as well as an optimal schedule for the objective function $\sum f_j(C_j)$ in which all machines process the orders in the same sequence.

(iii) If for some machine i there exists a machine h such that $p_{ij} \leq p_{hj}$ for $j = 1, \ldots, n$, then machine i does not play any role in determining the optimal schedule and may be ignored.

Some remarks with regard to these properties are in order. The second property does not hold for the more general problem in which the function $f_j(C_j)$ is not monotonic (e.g., problems that are subject to earliness and tardiness penalties). The third property is useful for reducing the size of an instance of a problem.

Consider the problem $PD \mid \beta \mid \sum f_j(C_j)$. Since this problem is strongly NP-hard, it is advantageous to develop dominance conditions or elimination criteria. We can prove the following order dominance result.

Lemma 2 *If in the problem $PD \mid\mid \sum f_j(C_j)$ there are two orders j and k such that $p_{ij} \leq p_{ik}$ for each $i = 1, 2, \ldots, m$, and*

$$\frac{\mathrm{d}f_j(t)}{\mathrm{d}t} \geq \frac{\mathrm{d}f_k(t)}{\mathrm{d}t}$$

then there exists an optimal schedule in which order j precedes order k.

Let us consider the total weighted completion time objective, $\sum w_j C_j$. Sung and Yoon (1998) showed the following result.

Theorem 3 *The problem $PD2 \mid\mid \sum w_j C_j$ is strongly NP-hard.*

Wagneur and Sriskandarajah (1993) considered the $\sum C_j$ objective. They presented a proof claiming that $PD2 \mid\mid \sum C_j$ is strongly NP-hard. Unfortunately, as pointed out by Leung *et al.* (2005), their proof is not correct. The

complexity status of this two machine problem remains open so far. However, Leung *et al.* (2002a) obtained the following result for three machines.

Theorem 4 *The problem* $PD3 \mid\mid \sum C_j$ *is NP-hard in the strong sense.*

Since the problem to minimise $\sum C_j$ is strongly NP-Hard with three or more machines, it makes sense to develop and analyse heuristics for this problem. A number of researchers have focused their attention on the development of heuristics and the following heuristics have been proposed for $PD \mid\mid \sum C_j$.

Definition 1 The Shortest Total Processing Time first (STPT) heuristic generates a sequence of orders one at a time, each time selecting as the next order the one with the smallest total amount of processing over all m machines.

Definition 2 The Shortest Maximum Processing Time first (SMPT) heuristic generates a sequence of orders one at a time, each time selecting as the next order the one with the smallest maximum amount of processing on any one of the m machines.

Definition 3 The Smallest Maximum Completion Time first (SMCT) heuristic first sequences the orders in nondecreasing order of p_{ij} on each machine $i = 1, 2, \ldots, m$, then computes the completion time for order j as $C_j' = \max_{i=1}^{m}\{C_{ij}\}$, and finally schedules the orders in nondecreasing order of C_j'.

Definition 4 The Shortest Processing Time first on the machine with the largest current load (SPTL) is a heuristic that generates a sequence of orders one at a time, each time selecting as the next order the one with the smallest processing time on the machine that currently has the largest load.

Definition 5 The Earliest Completion Time first (ECT) heuristic generates a sequence of orders one at a time; each time it selects as the next order the one that would be completed the earliest.

The STPT and the SMPT heuristics have been studied by Sung and Yoon (1998). They focused on two machines. Wang and Cheng (2003) studied the m machine case. Besides the STPT and the SMPT heuristics, they also analysed the SMCT heuristic. The last two heuristics, i.e. SPTL and ECT, were proposed by Leung *et al.* (2002a).

It turns out that all these heuristics may perform quite poorly in their worst case. For example, Wang and Cheng (2003) obtained the following worst-case bound.

Theorem 5 *For the problem* $PDm \mid\mid \sum C_j$,

$$\frac{\sum C_j(H)}{\sum C_j(OPT)} \leq m$$

where $H \in \{STPT, SMPT, SMCT\}$.

Leung *et al.* (2002a) showed that SPTL is unbounded. They also obtained the following result.

Theorem 6 *For the problem* $PDm \mid\mid \sum C_j$,

$$\frac{\sum C_j(ECT)}{\sum C_j(OPT)} \leq m$$

It is not clear whether the worst-case bound is tight for these heuristics. For two machines, Leung *et al.* (2002a) presented an instance for which the ratio is 1.618 for the STPT heuristic. They also gave an instance for which the ratio is $\sqrt{2}$ for both ECT and SMCT.

Leung *et al.* (2002a) performed an extensive empirical analysis showing that among the five heuristics described above, the ECT rule performs on the average clearly the best.

For minimising $\sum w_j C_j$, the performance bounds of the weighted version of STPT, SMPT, SMCT remain unchanged (Wang and Cheng, 2003). Leung *et al.* (2003a) also modified the ECT and SPTL heuristics to take the weights of the orders into account. The new heuristics are referred to as the WECT and WSPL rules. In detail, the WECT heuristic selects the next order j^* which satisfies

$$j^* = \arg\min_{j \in \Omega} \left\{ \frac{C_j - C_k}{w_j} \right\}$$

where C_k is the completion time of the order that was scheduled immediately before order j^*. A postprocessing procedure interchanges order j^* with order k in case $C_{j^*} \leq C_k$ in order to obtain a better (at least no worse) solution. Note that the case $C_{j^*} \leq C_k$ occurs only when $p_{i^*j^*} = 0$, where i^* is the machine on which order k has, over all machines, the largest finish time. Assume that after the swap the order immediately before order j^* is order l. If $C_{j^*} \leq C_l$, we proceed with an interchange of order j^* with order l. We continue with this postprocessing until C_{j^*} is larger than the completion time of the order that immediately precedes it. Note that after each swap, the finish time of j^* either decreases or remains unchanged, while the finish time of each order that is swapped with j^* remains unchanged. This is due to the fact that order j^* has zero processing time on the machine on which the swapped order has its largest finish time. Thus, the postprocessing, if any, produces a solution that is no worse than the one without postprocessing. Note that following the WECT heuristic, there may at times be ties. Since ties may be broken arbitrarily, the WECT heuristic may lead to various different schedules. For the performance of WECT, we can show the following result.

Theorem 7 *For the problem* $PDm \mid\mid \sum w_j C_j$,

$$\frac{\sum C_j(WECT)}{\sum C_j(OPT)} \leq m$$

The proof of the above theorem turns out to be much more complicated than that of Theorem 6. With additional constraints of processing times for each order, we can show that the performance bounds of the heuristics can be much better (close to 2 or 3). For details, see Leung *et al.* (2003a).

We also note that Wang and Cheng (2003) proposed an approximation algorithm which has a worst-case ratio of $16/3$. The algorithm is based on a linear programming relaxation which is formulated on the time intervals geometrically divided over the time horizon. Leung *et al.* (2003a) also presented a 2-approximation algorithm which is based on a linear programming formulation on the completion time of the orders. However, the implementation of this algorithm is quite complicated.

Leung *et al.* (2003a) did an extensive experimental analysis of the five simple heuristics listed above as well as of the $16/3$-approximation algorithm. Experimental results show that among the five simple heuristics, WECT is the best. For instances with certain characteristics, the WECT rule performs even better than the $16/3$-approximation algorithm.

The NP-hardness of problem $PD1 \mid \pi 1 \mid \sum T_j$ is a direct consequence of the NP-hardness of $1 \mid\mid \sum T_j$, see Du and Leung (1990).

The maximum lateness $L_{\max}$ can be minimised by the *Earliest Due Date* rule; i.e., the next finished product is assigned to the customer with the earliest due date. Through an adjacent pairwise interchange argument the following theorem can be shown.

Theorem 8 *The Earliest Due Date rule solves the problem* $PD \mid\mid L_{\max}$, *and the Preemptive Earliest Due Date rule solves the problem* $PD \mid prmt, r_j \mid L_{\max}$.

In a more general setting, the processing of the orders are subject to precedence constraints and the objective function is the more general $f_{\max}$. Lawler (1973) developed an algorithm based on (backwards) dynamic programming that solves the single machine version of this problem in polynomial time. Lawler's algorithm can be extended in such a way that it generates optimal solutions for the more general PD machine environment.

Theorem 9 *The problem* $PD \mid prec \mid f_{\max}$ *can be solved in* $O(n^2)$ *time.*

The problem $PD1 \mid r_j, \pi 1 \mid L_{\max}$ is NP-hard, which is a direct consequence of the NP-hardness of $1 \mid r_j \mid L_{\max}$.

The total number of late jobs objective, $\sum w_j U_j$, is also of interest. When $w_j = 1$, Wagneur and Sriskandarajah (1993) showed the following result.

Theorem 10 *The problem* $PD2 \mid\mid \sum U_j$ *is NP-hard in the ordinary sense.*

In fact, Cheng and Wang (1999) showed that there exists a pseudo-polynomial time algorithm for every fixed $m \geq 2$. When the number of machines is arbitrary, Ng *et al.* (2003) showed the following result.

Theorem 11 *The problem* $PD \mid\mid \sum U_j$ *is NP-hard in the strong sense. The NP-hardness in the strong sense remains in effect even for the very restricted case* $PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum U_j$.

For the problem $PD \mid d_j = d \mid \sum U_j$, Leung *et al.* (2002b) showed that the problem reduces to the Multiset Multicover (MSMC) problem (Rajagopalan and Vazirani, 1998). Thus, any algorithm solving the MSMC problem also solves the problem $PD \mid d_j = d \mid \sum U_j$. Leung, Li and Pinedo (2002b) adapted Rajagopalan and Vazirani's greedy algorithm for MSMC in such a way that it is also applicable to the $PD \mid d_j = d \mid \sum U_j$ problem. In the next theorem this modified version of the Rajagopalan and Vazirani algorithm is denoted by H_g. Leung *et al.* (2002b) obtained the following result for the H_g algorithm.

Theorem 12 *For* $PD \mid d_j = d \mid \sum U_j$, *if all* p_{ij} *and* d *are integers, then*

$$\frac{\sum U_j(H_g)}{\sum U_j(OPT)} \leq \mathcal{H}\left(\max_{1 \leq j \leq n}\left\{\sum_{i=1}^{m} \frac{p_{ij}}{c_i}\right\}\right)$$

where $c_i = GCD(p_{i1}, p_{i2}, \ldots, p_{in}, d)$ *for* $i = 1, 2, \ldots, m$, *and* $\mathcal{H}(k) \equiv \sum_{i=1}^{k}(1/i)$ *is the harmonic series. In addition, the bound is tight.*

It should be noted that Rajagopalan and Vazirani's greedy algorithm was designed for the weighted MSMC problem. If in our problem each order has a weight w_j, then it is also easy to adapt the greedy algorithm to solve $PD \mid d_j = d \mid \sum w_j U_j$. The approximation ratio of the revised greedy algorithm for $PD \mid d_j = d \mid \sum w_j U_j$ remains unchanged. Another observation for $PD \mid d_j = d \mid \sum w_j U_j$ is that it is in fact the dual problem of the multidimensional 0–1 knapsack problem (MKP) with an arbitrary number of dimensions. Thus, the resolution methods for MKP also shed light on solving $PD \mid d_j = d \mid \sum w_j U_j$. For a very recent survey for the MKP problem, the reader is referred to Fréville (2004).

For the restricted case $PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum U_j$, Ng *et al.* (2003) proposed a $(d+1)$-approximation algorithm based on a linear programming relaxation. However, if we apply the greedy algorithm for $PD \mid d_j = d \mid \sum U_j$ to solve $PD \mid p_{ij} \in \{0,1\}, d_j = d \mid \sum U_j$, the approximation ratio is at most $\mathcal{H}(m)$. Thus, if $m \leq e^d$, the approximation ratio of the greedy algorithm would be better than that of the LP-based heuristic. In fact, by our

reduction, the $PD \mid p_{ij} = 0 \text{ or } 1, d_j = d \mid \sum U_j$ problem turns out to be a set multicover (SMC) problem, since the elements in each constructed set are unique but each element requires to be covered multiple times. Thus, any approximation algorithm for the SMC problem can be applied to solve $PD \mid p_{ij} = 0 \text{ or } 1, d_j = d \mid \sum U_j$ with the approximation ratio being preserved. Hochbaum (1996) presented several LP-based ρ-approximation algorithms for weighted SMC, where

$$\rho = \max_{1 \le j \le n} \left\{ \sum_{i=1}^{m} p_{ij} \right\}$$

Clearly, these ρ-approximation algorithms can be applied to solve $PD \mid p_{ij} = 0 \text{ or } 1, d_j = d \mid \sum w_j U_j$. Since $\mathcal{H}(\rho) < \rho$ for $\rho \ge 2$, it is easy to see that the approximation ratio of our greedy algorithm is still better.

Heuristics for $PD \mid\mid \sum U_j$ can be designed based on the ideas of the Hodgson–Moore (1968) algorithm which solves the single machine version of this problem, i.e. $PD1 \mid \pi 1 \mid \sum U_j$, to optimality. The Hodgson–Moore algorithm for the single machine version generates a schedule by inserting the jobs in a forward manner one at a time according to the EDD rule. Whenever a job is completed after its due date, the procedure selects among the jobs that are currently still part of the schedule the longest one and takes it out of the schedule. Once a job has been taken out of the schedule, it never can get back in. Using the main idea behind the Hodgson–Moore algorithm the following heuristic can be designed for $PD \mid\mid \sum U_j$. The orders are put in the schedule S in a forward manner one at a time; whenever an order j' that is put into the schedule is completed after its due date, one of the orders that are currently part of the schedule has to be taken out.

The selection of the order that has to be taken out can be done based on a priority ranking system. In order to make sure that not more than one order has to be deleted from the schedule, it pays to keep a set of candidate orders S_c with the property that the removal of any one order in S_c from S ensures that the rest of the orders in S are completed before their due dates.

First of all, the tardy order j' itself is already a candidate order, since all the orders that precede j' in S can be completed before their due dates. For each order $j \in S, j \neq j'$, if its removal from S enables order j' to be completed in time, then j becomes a candidate in S_c, otherwise, j will not become a candidate. It is clear that $1 \le |S_c| \le |S|$.

Secondly, for each candidate order $j \in S_c$, a weighted sum of all its processing times p_{ij} on the m machines, denoted by $W(S_c)_j$, has to be computed. The weight of machine i, denoted by w_i, is a function of the current load on machine i, denoted by CL_i, and the future workload of machine i due to all the orders that still have to be considered, denoted by FL_i. A typical weight

function can be

$$w_i = \omega_1 CL_i + \omega_2 FL_i$$

where ω_1 and ω_2 are the weights for CL_i and FL_i, respectively, for any $i, 1 \leq i \leq m$. With w_i, the weighted sum of each candidate order $j \in S_c$ is computed as

$$W(S_c)_j = \sum_{i=1}^{m} w_i p_{ij}$$

Finally, the candidate order to be taken out is the one with the maximum weighted sum, i.e. order j^* such that

$$W(S_c)_{j^*} = \max_{j \in S_c} \Big(W(S_c)_j \Big)$$

Leung *et al.* (2002b) have done an extensive empirical analysis of the heuristic above. They also proposed an exact algorithm that uses constraint propagation, backtracking, and bounding techniques. Their result shows that the exact algorithm can solve instances of moderate size in a reasonable running time; and the results of the above heuristic are quite close to those of the exact algorithm. Recently, Leung *et al.* (2005) generalised the heuristic above so that it is applicable to the weighted case.

3. THE FULLY FLEXIBLE CASE

In the fully flexible case, the m machines are identical and each machine is capable of producing all k products. Two sets of problems are of interest, namely

(i) the fully flexible cases without setup times,

(ii) the fully flexible cases with arbitrary setup times.

Clearly, the fully flexible case is more difficult than the fully dedicated case, the reason being that we have to take care of two issues for this case: besides sequencing the orders, we need also to assign the product types to the machines. Recall that for the fully dedicated case, we need only to consider the issue of sequencing the orders.

3.1 The Fully Flexible Case Without Setup Times

The problem $PF1 \mid \pi k, \beta \mid \gamma$ is identical to the problem $1 \mid \beta \mid \gamma$. In the following we will consider $m \geq 2$ only. As we shall see, there are similarities as well as differences between the case $PFm \mid \pi k, \beta \mid \gamma$ and the standard parallel machine environment $Pm \mid \beta \mid \gamma$.

For the objectives of $C_{\max}$, $L_{\max}$ and $\sum w_j T_j$, the complexities follow closely those of the standard parallel machine scheduling environment. Thus,

$$PF \mid prmt, \pi k \mid C_{\max} \quad \text{and} \quad PF \mid prmt, \pi k \mid L_{\max}$$

are solvable in polynomial time, whereas

$$PF2 \mid \pi 1 \mid C_{\max}, \; PF2 \mid \pi 1 \mid L_{\max} \quad \text{and} \quad PF2 \mid \pi 1 \mid \sum T_j$$

are NP-hard (since $PF1 \mid \pi 1 \mid \sum T_j$ is NP-hard).

On the other hand, the complexities are different for the $\sum C_j$ objective. It is well known that $P \mid\mid \sum C_j$ and $P \mid prmt \mid \sum C_j$ can be solved by the Shortest Processing Time first (SPT) rule (Pinedo, 2002). The following result was obtained by Blocher and Chhajed (1996).

Theorem 13 *The problem $PF2 \mid \pi k \mid \sum C_j$ is NP-hard in the ordinary sense.*

When $k = 2$, Yang (1998) showed that the problem remains NP-hard.

Theorem 14 *The problem $PF2 \mid \pi 2 \mid \sum C_j$ is NP-hard in the ordinary sense.*

But it is not known whether or not there exists a pseudo-polynomial time algorithm for the above ordinary NP-hard problem. When the number of machines is arbitrary, Blocher and Chhajed (1996) showed the following result.

Theorem 15 *The problem $PF \mid \pi k \mid \sum C_j$ is NP-hard in the strong sense.*

Consider now the preemptive version of this same problem. One can think of two different sets of assumptions for a preemptive version of this problem.

Under the first set of assumptions, the processing of a particular product type for a given order may be shared by multiple machines and the various machines are allowed to do this processing simultaneously. It is easy to show that the class of problems in this case, i.e. $PF \mid prmt, \beta \mid \gamma$ is identical to the class of problems $1 \mid prmt, \beta \mid \gamma$. This equivalence implies that this particular preemptive version is very easy.

Under the second set of assumptions for a preemptive version of this problem, any product for any given order can be processed partly on one machine and partly on an other. However, now we assume that if a product type for a given order is done on more than one machine, then these different processing times are not allowed to overlap. Leung *et al.* (2002c) showed that this particular preemptive version is hard.

Theorem 16 *The problem* $PF2 \mid prmt, \pi 2 \mid \sum C_j$ *is NP-hard in the ordinary sense.*

It is possible to design for $PF \mid \pi k \mid \sum C_j$ heuristics that have two phases. The first phase determines the sequence of the orders, and the second phase assigns the different products of an order to the machines. Based on these ideas, Blocher and Chhajed (1996) developed two classes of heuristics.

The first class of heuristics can be referred to as the *static two phase heuristics*. In these two phase heuristics, the orders are sequenced first, and then the different products for each order are assigned to the machines. Rules for sequencing the orders include:

- The *smallest average processing time first* (SAPT) rule sequences the orders in increasing order of $\sum_{l=1}^{k} p_{lj}/m$.
- The *smallest maximum completion time first* (SMCT) rule sequences the orders in increasing order of $C_{LPT}^{(j)}, j = 1, 2, \ldots, n$, where $C_{LPT}^{(j)}$ is the makespan of the schedule that is obtained by scheduling the different product types of order j on the m parallel machines according to the *longest processing time first* (LPT) rule, assuming each machine is available from time zero on.

After the sequence of orders is determined by one of the above rules, the product types of each order can be assigned to machines following one of the two assignment rules below:

- The *Longest Processing Time first* rule (LPT) assigns in each iteration an unassigned product type with the longest processing time to a machine with the smallest workload, until all product types are scheduled.
- The *Bin Packing* rule (BIN) starts by determining the completion time of an order using the LPT assignment rule above (just as a trial, not being the real assignment). This completion time is used as a target completion time (bin size). In each iteration, the BIN rule assigns an unassigned product type with the longest processing time to one of the machines with the largest workload. If the workload of the machine exceeds the target completion time after the assignment, then undo this assignment and try the assignment on the machine with the second largest workload. This procedure is repeated until the product type can be assigned to a machine without exceeding the target completion time. If assigning the product type to the machine with the smallest workload still exceeds the target completion time, then assign it to this machine, and reset the target completion time to the completion time of the product type on this machine.

Combinations of the sequencing rules with the assignment rules lead to four different heuristics: namely, SAPT-LPT, SAPT-BIN, SMCT-LPT, and SMCT-BIN.

The second class of heuristics may be referred to as the *dynamic two-phase heuristics*. In these heuristics, the sequence of orders is not fixed prior to the assignment of product types to machines, i.e., the sequence is determined dynamically. The heuristics still use the LPT rule or the BIN rule for the assignment. However, to determine the next order to be sequenced, a greedy approach is applied to make a trial assignment of the product types of all remaining orders by using either the LPT or the BIN rule, and the order that gives the smallest completion time is selected as the next order in the sequence. These two heuristics may be referred to as *Greedy-LPT* and *Greedy-BIN*.

Blocher and Chhajed (1996) did not obtain performance bounds for these heuristics. However, they did an experimental analysis and found that the results obtained with the heuristics are very close to the lower bound they developed. They also found that not one of the heuristics consistently dominates any one of the others. It turns out that these heuristics do have certain performance bounds. For details, the reader is referred to Leung *et al.* (2003b).

3.2 The Fully Flexible Case With Setup Times

The scheduling problems become considerably harder with arbitrary setup times, even for a single machine. Leung *et al.* (2003b) considered $PF1 \mid s_l \mid L_{\max}$ and $PF1 \mid s_l \mid \sum U_j$ and proved the following theorem.

Theorem 17 *The problems* $PF1 \mid s_l \mid L_{\max}$ *and* $PF1 \mid s_l \mid \sum U_j$ *are both NP-hard in the ordinary sense.*

This result is rather surprising, since $1 \mid\mid L_{\max}$ can be solved by the Earliest Due Date rule and $1 \mid\mid \sum U_j$ can be solved by the Hodgson–Moore algorithm.

Minimising $C_{\max}$ on one machine is solvable in polynomial time. All we need to do is to batch all requests of the same product type together and sequence the product types in an arbitrary order. In this way, each product type will incur its setup time at most once and $C_{\max}$ will be minimised. Unfortunately, Leung *et al.* (2003b) showed that we lose polynomiality when we have two or more machines.

Theorem 18 *The problems* $PF2 \mid s_l \mid C_{\max}$ *and* $PF2 \mid prmt, s_l \mid C_{\max}$ *are both NP-hard in the ordinary sense.*

For the objective of minimising the total completion time, some work has been done recently for the single machine case. Ng *et al.* (2002) showed the following result.

Theorem 19 *The problem* $PF1 \mid s, \pi k, p_{lj} \in \{0,1\} \mid \sum C_j$ *is strongly NP-hard.*

Interestingly, they mentioned that the complexity of $PF1 \mid s_l, \pi k, p_{lj} > 0 \mid \sum C_j$ remains an open problem. If there is a requirement for all the operations of any given product type to be scheduled contiguously (i.e. the operations for each product type must be scheduled in one batch), then the process is said to follow the *group technology* (GT) approach (see Gerodimos *et al.*, 1999; Ng *et al.*, 2002). While Gerodimos *et al.* (1999) showed that $PF1 \mid s_l, \pi k, GT, p_{lj} > 0 \mid \sum C_j$ is polynomially solvable, Ng *et al.* (2002) obtained the following complexity result.

Theorem 20 *The problem* $PF1 \mid s, \pi k, GT, p_{lj} \in \{0,1\} \mid \sum C_j$ *is strongly NP-hard.*

For two machines, we can derive the following result from Theorem 16.

Theorem 21 *The problem* $PF2 \mid prmt, s, \pi k \mid \sum C_j$ *is NP-hard in the ordinary sense.*

Consider now the case in which the orders have different release dates. Leung *et al.* (2003b) showed that no online algorithm (i.e. algorithms that operate without any knowledge of future order arrivals) can do as well as an optimal offline algorithm.

Theorem 22 *There is no optimal online algorithm for* $PF1 \mid r_j, s, \pi k \mid C_{\max}$ *and* $PF1 \mid prmt, r_j, s, \pi k \mid C_{\max}$.

4. CONCLUDING REMARKS

Order scheduling models have many real world applications. It is a relatively new idea to optimise the objective functions of the completion times of orders rather than individual completion time of the jobs included in the orders. Some work has been done in the past for special cases. Our goal has been to classify the previous work in a single framework based on the characteristics of machine environment and objectives. While the complexity of some of the problems are known, many remain open. It will be interesting to settle these issues in the future. Developing heuristics with provably good worst case bounds and/or good empirical performance is also a worthwhile direction to pursue.

References

Blocher, J. and Chhajed, D. (1996) The customer order lead-time problem on parallel machines. *Naval Research Logistics*, **43**:629–654.

Brucker, P. (1995) *Scheduling Algorithms*. Springer, Berlin.

Cheng, T. and Wang, G. (1999) Customer order scheduling on multiple facilities. *Technical Report* 11/98-9, Faculty of Business and Information Systems, The Hong Kong Polytechnic University.

Du, J. and Leung, J. Y.-T. (1990) Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, **15**:483–495.

Duenyas, I. (1994) Estimating the throughput of a cyclic assembly system. *International Journal of Production Research*, **32**:403–1410.

Fréville, A. (2004) The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, **155**:1–21.

Gerodimos, A., Potts, C., and Tautenhahn, T. (1999) Scheduling multi-operation jobs on a single machine. *Annals of Operations Research*, **92**:87–105.

Graham, R., Lawler, E., Lenstra, J., and Rinnooy Kan, A. (1979) Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, **5**:287–326.

Hochbaum, D. (1996) Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. In *Approximation Algorithms for NP-Hard Problems*, D. Hochbaum (Ed.), PWS Publishing Company, Boston, MA, pp. 94–143.

Julien, F. and Magazine, M. (1990) Scheduling customer orders—an alternative production scheduling approach. *Journal of Manufacturing and Operations Management*, **3**:177–199.

Lawler, E. (1973) Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, **19**:544–546.

Lee, C., Cheng, T., and Lin, B. (1993) Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. *Management Science*, **39**:616–625.

Leung, J. Y.-T., Li, H. and Pinedo, M. (2002a) Order scheduling in an environment with dedicated resources in parallel. *Journal of Scheduling*, accepted for publication.

Leung, J. Y.-T., Li, H., and Pinedo, M. (2002b) Scheduling orders for multiple product types with due date related objectives. *European Journal of Operational Research*. Accepted for publication.

Leung, J. Y.-T., Lee, C. Y., Young, G. H. and Ng, C. W. (2002c) Minimizing total flow time in generalized task systems. Submitted.

Leung, J. Y.-T., Li, H., Pinedo, M. and Sriskandarajah, C. (2005) Open shops with jobs overlap—revisited. *European Journal of Operational Research*. Accepted for publication.

Leung, J. Y.-T., Li, H., and Pinedo, M. (2003b) Order scheduling in a flexible environment with parallel resources. *Working Paper*.

Leung, J. Y.-T., Li, H., and Pinedo, M. (2003c) Scheduling multiple product types with weighted objectives. *Working Paper*.

Leung, J. Y.-T., Li, H., and Pinedo, M. (2003a) Scheduling orders for multiple product types to minimize total weighted completion time. *Working Paper*.

Moore, J. (1968) An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, **15**:102–109.

Ng, C., Cheng, T., and Yuan, J. (2002) Strong NP-hardness of the single machine multi-operation jobs total completion time scheduling problem. *Information Processing Letters*, **82**:187–191.

Ng, C., Cheng, T., and Yuan, J. (2003) Concurrent open shop scheduling to minimize the weighted number of tardy jobs. *Journal of Scheduling*, **6**:405–412.

Pinedo, M. (2002) *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Englewood Cliffs, NJ.

Potts, C., Sevast'janov, S., Strusevich, V., Wassenhove, L., and Zwaneveld, C. (1995) The two-stage assembly scheduling problem: complexity and approximation. *Operations Research*, **43**:346–355.

Rajagopalan, S. and Vazirani, V. (1998) Primal–dual rnc approximation algorithms for set cover and covering integer programs. *SIAM Journal on Computing*, **28**(2):525–540.

Sung, C. and Yoon, S. (1998) Minimizing total weighted completion time at a pre-assembly stage composed of two feeding machines. *International Journal of Production Economics*, **54**:247–255.

Wagneur, E. and Sriskandarajah, C. (1993) Open shops with jobs overlap. *European Journal of Operational Research*, **71**:366–378.

Wang, G. and Cheng, T. (2003) Customer order scheduling to minimize total weighted completion time. In *Proceedings of the 1st Multidisciplinary Conference on Scheduling Theory and Applications*, pp. 409–416.

Yang, J. (1998) Scheduling with batch objectives. *Ph.D. Thesis*, Industrial and Systems Engineering Graduate Program, The Ohio State University, Columbus, OH.

Multi-criteria Scheduling