Taylor & Francis
Taylor & Francis Group

Production Planning & Control
The Management of Operations

ISSN: 0953-7287 (Print) 1366-5871 (Online) Journal homepage: https://www.tandfonline.com/loi/tppc20

A branch and bound algorithm to minimise total weighted tardiness on a single batch processing machine with ready times and incompatible job families

S. K. Tangudu & M. E. Kurz

To cite this article: S. K. Tangudu & M. E. Kurz (2006) A branch and bound algorithm to minimise total weighted tardiness on a single batch processing machine with ready times and incompatible job families, Production Planning & Control, 17:7, 728-741, DOI: 10.1080/09537280600901467

To link to this article: https://doi.org/10.1080/09537280600901467

Production Planning & Control,
Vol. 17, No. 7, October 2006, 728–741

Taylor & Francis
Taylor & Francis Group

A branch and bound algorithm to minimise total weighted tardiness on a single batch processing machine with ready times and incompatible job families

S. K. TANGUDU and M. E. KURZ*

Department of Industrial Engineering, Clemson University, Clemson, SC 29634-0920

This research deals with developing a branch and bound algorithm for minimising total weighted tardiness on a single batch processing machine with characteristics similar to those seen in the diffusion operation in semiconductor manufacturing. Dominance properties are utilised in order to increase the efficiency of the algorithm. The developed algorithm is implemented in C++ and tested on different cases. We find the algorithm is capable of solving problems with up to 32 jobs. Computational results are presented along with a discussion of the effectiveness of the proposed algorithm and the effects of various parameters on the efficiency of the algorithm. The branch and bound algorithm can be used to solve relatively small problems as well as provide a means by which researchers can evaluate the effectiveness of their heuristics.

Keywords: Wafer production; Batch processing; Scheduling; Branch and bound

1. Introduction

Batch processing is defined as the processing of a number of jobs simultaneously. A few advantages of batch processing are that the jobs in the batch get the same treatment environment, resulting in identical characteristics for that stage of processing, as well as higher utilisation of resources. Wafer fabrication uses batch processing and is one of the most complex production processes in industry. Complex manufactured products, such as wafer fabrication in the semiconductor industry, require many steps, several of which may utilise batch processing. Individual batch processing operations in wafer fabrication can have processing times of the order of 10 hours, and dyeing and printing operations in the textile industry can have processing times of the order of 6 to 12 hours. Such large processing times can have profound effects on the overall production rate. The complexity of the processes makes it a challenging task to fulfil orders on time. In real world scenarios, manufacturers face the situation of frequently not being able to fulfil orders on time and jobs which may arrive for processing at different times. Careful batch formation and batch sequencing becomes important in batch processing to minimise total weighted tardiness; as Azizoglu and Webster (2000) point out in the context of the burn-in operation in the semi-conductor industry, there is a conflict between utilising the most space in a batch and ensuring that more valued jobs are processed earlier.

We examine the problem of scheduling a set of n jobs to minimise total weighted tardiness (TWT) on a single batch processing machine which can process up to B jobs simultaneously. B is defined as the maximum batch size. Each job has a known non-zero ready time r_j, so the problem is considered dynamic (as opposed to static,

*Corresponding author. Email: mkurz@clemson.edu

Production Planning and Control
ISSN 0953–7287 print/ISSN 1366–5871 online © 2006 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/09537280600901467

where all jobs are ready at time zero). Each job j also has a due date d_i and a weight w_j, which reflects the importance of the job. It also reflects a linear penalty assessed for each time unit that the job is completed after its due date. Jobs belong to one of f families. Processing times of jobs in the same family are equal and jobs from different families may not be processed in the same batch; jobs with this condition are often said to belong to incompatible families. The machine is available for continuous processing. Once processing of a batch is initiated, it cannot be interrupted. The ready time of a batch is given by the longest ready time of all jobs in the batch. Scheduling a single batch processing machine consists of forming batches and sequencing these batches. The sequencing of batches is independent (i.e. set-up times are not sequence dependent). These characteristics echo those seen in the diffusion operation in the semi-conductor industry. This problem is NP-hard based on reductions to one of two known NP-hard problems which include characteristics of other known NP-hard problems: the single machine total tardiness problem with ready times (Du and Leung 1990) and the single batch processing machine total weighted tardiness problem with incompatible jobs (Perez *et al*. 2005).

2. Literature review

Batch processing machines differ from (regular) unit processing machines in that more than one job can be processed simultaneously on the machine at the same time. This relaxes one of the most important assumptions in most scheduling literature (Rinnooy Kan 1976). A subset of jobs is processed simultaneously in a batch; the maximum number of jobs that can be in a batch is called the 'maximum batch' size (B) and the actual number of jobs assigned to a batch is simply called the 'batch' size. In order to better describe the types of batch processing that are addressed in literature, we will use 'p-batch'when the processing time of the batch depends only on the type of jobs that are in the batch, not the number of jobs that are in the batch (Brucker and Knust 2006). One example of general p-batch processing comes from curing ovens at factories that make electronic control units for anti-lock brakes—all parts are cured simultaneously. We are interested in assigning jobs to batches and sequencing batches on machines, including determining the completion time of each batch on each machine. We refer to this process as 'scheduling'.

We use a modified version of the notation introduced by Graham *et al*. (1979) and extended by many researchers including Brucker and Knust (2003). In this paper, we use the following additional components: p-batch, indicating p-batch processing as described above; incompat, indicating incompatible families; and a_j when each job j takes up an amount of space a_j in the batch processing machine. We denote the problem of interest as

$$1|p-batch, r_j, incompat|\sum w_j T_j.$$

The literature review summarises recent papers dealing with scheduling on a single p-batch processing machine as well as papers dealing with the TWT objective for parallel machines. The single p-batch processing machine papers are generally motivated by the burn-in operation in semiconductor manufacturing. In the burn-in operation, different wafers may have different processing times but may be placed in the same batch; the batch processing time is the maximum of the individual wafer processing times (Uzsoy 1994). Because jobs with different processing times can be in the same batch (compatible families) in the burn-in operation, we will consider this to be the default configuration for families and will only indicate when this does not hold. Webster and Baker (1995) include several results for dynamic p-batch scheduling, focusing on the objective functions of makespan, total flowtime and maximum lateness. Uzsoy (1995) succinctly reviews most of the related literature up to 1994. Due to space constraints, we refer the interested reader to these papers and focus this review on literature appearing after 1994 and one specific reference appearing in 1993. We organise this brief literature review by the objective function considered.

The makespan objective function has been recently considered by numerous researchers. Sung and Choung (2000) provide an integer programming (IP) formulation. Optimal algorithms are given for the static case while several solution methods, including branch and bound (B&B), dynamic programming (DP) and heuristics based on the optimal methods for the static case, are given for the dynamic case. Sung *et al*. (2002) consider the dynamic case where jobs belong to one of a fixed number of compatible families with a DP algorithm. Dupont and Dhaenens-Flipo (2002) develop a B&B algorithm for $1|p-batch, a_j|C\max$, solving some problems with up to 100 jobs. Cheraghi *et al*. (2003) consider jobs that all have the same processing time, which is overly restrictive for research, motivated by the burn-in operation, and must complete no later than their due dates, developing a non-linear mixed integer programming (MIP) formulation and implementing a genetic algorithm.

Minimising maximum tardiness and the number of tardy jobs in dynamic batch scheduling, motivated by the burn-in operation, is considered by Li and Lee (1997).

They prove that even when ready times and due dates are agreeable ($r_i < r_j \Rightarrow d_i \leq d_j$), the problem of finding the optimal solution for either objective is NP-hard in the strong sense. They provide DP algorithms for both objective functions when ready times, due dates and processing times are all agreeable. Without the restriction of agreeable problem data, Wang and Uzsoy (2002) consider minimising maximum lateness in presence of ready times, which is also strongly NP-hard, with solution methods based on DP, heuristics and an adaptation of the random keys genetic algorithm.

$1|p-batch|\sum C_j$ has been considered by Dupont and Ghazvini (1997), who used a B&B algorithm, and Hochbaum and Landy (1997), who developed an optimal polynomial time heuristic for this problem. Dupont and Ghazvini (1997) are able to solve problems with up to 100 jobs. Hochbaum and Landy (1997) compare their results to the B&B algorithm developed by Chandru *et al.* (1993). Implementing Chandru *et al.*'s B&B algorithm for the same problem, Hochbaum and Landy are not able to solve problems with 100 jobs, while their polynomial time algorithm is able to solve problems of up to 10 000 jobs. Hochbaum and Landy (1997) also provide a heuristic solution for the problem with identical parallel batch processing machines.

Minimising total earliness and tardiness, which is not a regular performance measure, is considered by Qi and Tu (1999). In their research, all jobs have the same processing time, as in Cheraghi *et al.* (2003). They consider several variants of the problem, including weighted earliness and tardiness, agreeable ready times and due dates, and due date windows.

Uzsoy and Yang (1997) compared a B&B algorithm with several heuristics for $1|p-batch|\sum w_jC_j$, solving problems of up to 20 jobs with the B&B algorithm and solving problems of up to 100 jobs with the heuristics. Azizoglu and Webster (2000) considered the same problem generalised to $1|p-batch, a_j|\sum w_jC_j$, developing another B&B algorithm which solves some problems of up to 25 jobs. Azizolgu and Webster (2001) also considered $1|p-batch, a_j, incompat|\Sigma w_jC_j$, a problem more like the one of interest in this research, developing a B&B algorithm that solves some problems with up to 25 jobs.

The authors are aware of only a few papers focusing on the total tardiness or TWT objectives for a single p-batch machine. Mehta and Uzsoy (1998) show that $1|p-batch,\ incompat|\Sigma T_j$ is strongly NP-hard. A DP solves problems with up to 360 jobs and several heuristics are developed, which solve the problems, though not necessarily to optimality, much more quickly. Perez *et al.* (2005) address $1|p-batch, incompat|\Sigma w_jT_j$ through the use dispatching rules. Kurz and Mason (2005) develop a batch improvement algorithm which is applied to $1|r_j, p-batch, incompat|\Sigma w_jT_j$, solving problems with up to 360 jobs. This is the only known paper that primarily considers the exact problem of interest in this paper (Perez *et al.* 2005 do not include ready times). Closely related but focused on parallel p-batch processing machines are Kurz (2003), Balasubramanian *et al.* (2004) and Mönch *et al.* (2005). Kurz (2003) provides some results on the structure of an optimal schedule for $P|p-batch, r_j|\Sigma w_jT_j$ with a single family. Balasubramanian *et al.* (2004) consider $P|p-batch, incompat|\Sigma w_jT_j$, developing several heuristics and genetic algorithms. Mönch *et al.* (2005) consider $Pm|r_j, p-batch,\ incompat|\Sigma w_jT_j$ and address $1|r_j, p-batch,\ incompatible|\Sigma w_jT_j$ as a sub-problem in which the single machine schedule is created using a constructive algorithm. All of these papers, except Kurz (2003), consider an environment in which the jobs in a batch must belong to the same family, such as in the diffusion operation in the semi-conductor industry. Potts and Kovalyov (2000) consider both serial batch (in the sense of lot-sizing) and p-batch scheduling in an extensive literature review, though no papers in that review include the characteristic of incompatible job families. The only paper considering the TWT objective listed in Potts and Kovalyov (2000) is by Brucker *et al.* (1998) in which neither ready times nor incompatible families are considered. Mathirajan and Sivakumar (2003) provide a focused review of batch processing scheduling in the semiconductor manufacturing industry, providing insight into the recent historical interest in the topic. The authors are unaware of any papers in the open literature reporting optimal solution methods for $1|p-batch, r_j, incompatible|\Sigma w_jT_j$, which we desire in order to facilitate the evaluation of heuristic and meta-heuristic solution methods in future research.

3. Proposed branch and bound algorithm

In this section a branch and bound (B&B) algorithm is proposed. We begin with the overall algorithm structure, which is somewhat similar to that used in Chandru *et al.* (1993). The proposed B&B algorithm utilises an upper bound (UB) and a lower bound (LB) at every node; this will be discussed below. The fathoming and branching decisions are also discussed. We finish the section with an example.

3.1 *Overall algorithm structure*

Initially, an overall upper bound (UB) on the TWT for a problem instance must be identified. The initial value of UB can be the TWT of any feasible solution to the

problem, or it may be set to be large enough that any feasible solution will have a lower TWT. Once the UB's value has been determined, we begin branching from the current node. In the initial stage of the algorithm, all jobs are sorted in increasing order of ready time. In the proposed algorithm, we will implement a depth-first strategy, based on our desire to more quickly find a complete feasible schedule Furthermore, we anticipate experimentation with a truncated version of the proposed B&B algorithm for $1|p-batch, r_j, incompat|\Sigma w_j T_j$ as a sub-procedure in a heuristic solution method for $P|p-batch, r_j, incompat|\Sigma w_j T_j$.

Each node is defined by a partial sequence σ of jobs partitioned into batches which are scheduled consecutively from the beginning of the schedule. For each active node, child nodes are created by appending batches to the end of the current partial sequence. The batches to be appended are created by forming all possible batches of unscheduled jobs with size ranging from 1 to the smaller of B and the number of remaining unscheduled jobs, ensuring that only jobs of the same family are placed into the same batch. Without loss of generality, consider the case of family f which has $n_f \geq B$ unscheduled jobs. We have

$$\sum_{j=1}^{f} \binom{n_f}{f}$$

possible child nodes for family f. Clearly, as n_f increases and the number of families increase, the total number of child nodes increases greatly as well; it will be desirable to fathom nodes as early as possible in order to prune the search tree quickly. This paper includes a dominance property which will allow us to greatly reduce the number of possible child nodes at each level.

After a child node is formed we compute a lower bound (LB), which is described in a later section. If the child node does not represent a complete solution, we compare LB to UB and fathom the node if LB > UB. If the child node represents a complete solution, we compute the TWT. If the resulting value is lower than UB, we replace UB by the resulting value and the current node becomes the incumbent solution.

When all nodes have been either fathomed or identified as having the same TWT as the incumbent solution, we have completed the application of the B&B algorithm and have found an optimal solution to the problem.

3.2 *Upper bound calculation*

We may use a sufficiently large number or the TWT of a feasible solution as the initial upper bound at the root node. In general, the lower the active upper bound, the faster a B&B algorithm can determine the optimal solution to the problem at hand. Therefore, in the experimental study to be discussed in a later section, we have utilised the batch improvement algorithm (BIA) by Kurz and Mason (2005) to form an initial feasible solution. For completeness, we summarise this method here in Appendix A while noting that it is not in itself part of this research.

BIA considers a current solution and iterates from the last to the first batch. BIA tries smartly to move jobs from later batches to the current batch without increasing the starting time of the current batch. Initially, batches are formed greedily by considering the jobs sorted in non-decreasing order of ready times, with ties being broken by non-decreasing order of weighted due date. This initial job sequencing is made without regard to job family designations. Batches of jobs in the same family are then formed by adding jobs one at a time according to the sorted list until either (1) the current batch is full (i.e. B same-family jobs have been grouped together) or (2) a different job family is encountered, thereby forcing the formation of a new batch, given our incompatible job families assumption.

In step 1, we determine whether the current batch needs any jobs added to it. Step 2 begins the search for jobs to add to the current batch by first determining to which family the additional jobs should belong. Once we complete step 2, we have a batch that has space for at least one job and we know the family to which the additional jobs should belong.

In step 3, we determine which jobs can be added to the current batch. In step 4, we add jobs to the current batch as long as the current batch has room and there are eligible jobs. Each batch which has jobs removed from it must also have BIA performed upon it. Once step 4 is complete, we consider the next earlier batch and begin BIA on that batch.

3.3 *Lower bound computations*

For the purposes of this section, we assume that a partial solution σ is given. LB(σ) has two components – one for the scheduled jobs and one for the unscheduled jobs – which will be discussed below. Let $\mathbf{S}(\mathbf{U})$ be the set of (un)scheduled jobs and LB(σ) be the corresponding lower bound at the current node. We have $\mathrm{LB}(\sigma) = \mathrm{LB}(\mathbf{S}) + \mathrm{LB}(\mathbf{U})$.

At each node, the current partial solution will be expanded by appending one or more jobs in a batch. With this child creation methodology, we will never change the existing batches. Therefore, LB($\mathbf{S}$) is the true value of the TWT for the scheduled jobs at the current node. We compute the TWT for the scheduled jobs as follows. Let $J^{[k]}(\sigma)$ be the set of jobs in batch k from a

partial schedule σ. All jobs in the same family require the same processing time $P^{[k]}(\sigma)$ and only jobs in the same family can be assigned to the same batch. Let $R^{[k]}(\sigma)$ be the ready time of batch k in σ, which is the maximum of the ready times for all jobs in the batch; $R^{[k]}(\sigma) = \max_{i \in J^{[k]}(\sigma)}\{r_i\}$. Let $C^{[k]}(\sigma)$ be the completion time of batch k. All jobs in a batch have the same completion time which is the sum of the batch start time and the batch processing time; the batch start time is the larger of its ready time and the completion time of the previous batch: $C^{[k]}(\sigma) = \max\{C^{[k-1]}(\sigma), R^{[k]}(\sigma)\} + P^{[k]}(\sigma)$ and $C_i^{[k]}(P) = C^{[k]}(P)$, $\forall i \in J^{[k]}(P)$. We then compute the TWT of the scheduled jobs by summing the individual weighted tardiness of the scheduled jobs, resulting in $\mathrm{LB}(\mathbf{S}) = \sum_{i \in \mathbf{S}} w_i \max(0, C_i(\sigma) - d_i)$.

When computing LB(**U**), we use modified ready times for each unscheduled job. Job j's modified ready time is the larger of its ready time and the completion time of the last batch of σ; this can be expressed as the maximum of the job's ready time and the completion time of all jobs in **S** so $r_j' = \max(r_j, \max_{j \in S}\{C_j(\sigma)\})$. We utilise a simple lower bound which essentially assumes that all jobs can be processed in parallel: $\mathrm{LB}(\mathbf{U}) = \sum_{j \in \mathbf{U}} w_j \max(0, r_j' + p_j - d_j)$.

3.4 *Dominance properties*

For completeness, we repeat three results: Proposition 2 from Mehta and Uzsoy (1998) (called MU Proposition 2) concerning the form of an optimal schedule for $1|p-batch, incompat|\gamma$, where γ is a regular measure of performance (which TWT is); and Proposition 2 and Lemma 3 from Uzsoy (1995) (called U Proposition 2 and U Lemma 3) concerning the form of an optimal schedule for $1|p-batch, incompat|\Sigma w_j C_j$.

MU Proposition 2: *In a static* [*emphasis added*] *batch processing machine scheduling problem with incompatible job families and a regular measure of performance, there exists an optimal schedule which contains no partially full batches except possibly the last batch of each family to be processed in the schedule.*

[*Note*: MU Proposition 2 assumes that jobs in the same family are indexed in non-decreasing order of due dates.]

U Proposition 2: *Consider the static* [*emphasis added*] *batch processing machine scheduling problem with incompatible job families where the performance measure to be minimised is total weighted completion time. Let us index the jobs in each family in decreasing order of their weight. Then there exists an optimal schedule where all batches of the same family contain consecutively indexed jobs.*

U Lemma 3: *Consider the problem of minimising total weighted completion time on a single or parallel identical batch processing machines with incompatible families. There exists an optimal schedule where all batches processed on the same machine are sequenced in increasing order of p^k/W_k, where p^k is the processing time of the family to which the jobs belong and W_k is the sum of the job weights of the jobs in the batch.*

While these were developed for static problems, we can utilise them in some parts of our solution procedure for dynamic problems, based on our Proposition 1. In Proposition 1, we see that, after a partial sequence σ has been created and certain conditions are met, the scheduling of the remaining jobs may be done optimally. We use the decreasing order of $\mathrm{W_k/p^k}$ to emphasise a relationship with the WSPT (weighted shortest processing time) rule.

Proposition 1: *Consider $1|p-batch, r_j, incompat|\sum w_j T_j$ with a partial sequence σ of jobs partitioned into batches, which are scheduled consecutively from the beginning of the schedule. If $r_j \le \max_{i \in \mathbf{S}}\{C_i(\sigma)\}$ and $d_j \le \max_{i \in \mathbf{S}}\{C_i(\sigma)\}, \forall j \in \mathbf{U}$, then there exists an optimal completion of the partial sequence with the following characteristics*:

(i) *The optimal completion contains no partially full batches except possibly the last batch of each family to be processed in the schedule.*
(ii) *The batches are formed, after indexing the jobs in each family in decreasing order of their weight, by considering consecutively indexed jobs.*
(iii) *The batches are sequenced in decreasing order of W_k/p^k.*

Proof: Once the jobs in **U** have ready times no later than $\max_{i \in \mathbf{S}}\{C_i(\sigma)\}$, the problem reduces to $1|p-batch, incompat|\Sigma w_j T_j$ for the jobs in **U**, and so MU Proposition 2 applies. When the jobs in **U** also have due dates earlier than $\max_{i \in \mathbf{S}}\{C_i(\sigma)\}$ the problem reduces to $1|p-batch, incompact|\Sigma w_j C_j$ and U Proposition 2 and U Lemma 3 hold.

Proposition 1 can be used to create Dominance Property 1.

Dominance Property 1: Consider a node representing a partial sequence σ. If the last scheduled batch completes at time $\max_{i \in \mathbf{S}}\{C_i(\sigma)\}$ and $r_j \le \max_{i \in \mathbf{S}}\{C_i(\sigma)\}$ and $d_j \le \max_{i \in \mathbf{S}}\{C_i(\sigma)\}$, $\forall j \in \mathbf{U}$, then an optimal completion can be found by applying the following algorithm:

1. Sort the jobs in each family in decreasing order of their weights.
2. For each family, form batches by grouping B jobs at a time taken consecutively from the sorted list created in step 1, where the last batch of each family may possibly have less than B jobs.

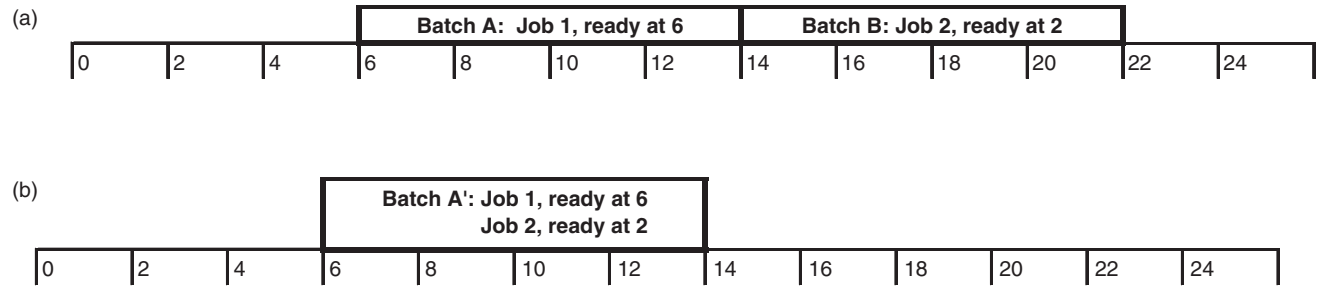

Figure 1. (a) Two batches before Proposition 2 applied. (b) One batch after Proposition 2 applied.

3. Sort the batches in decreasing order of $W_k/p^k$and schedule the batches in that order on the single machine.

Theorem 1 of Kurz (2003) (called K Theorem 1) is restated here. K Theorem 1 was written to address $P|p-batch, r_j|\sum w_j T_j$. It basically says that if a job is ready before the batch before the one it is in, and that batch has room, the TWT will not be increased by moving the job into the earlier partially full batch. In our Proposition 2, we simply indicate that K Theorem 1 can be applied individually to the batches of jobs from the same family on a single machine. We utilise the notation introduced in section 3.3.

K Theorem 1: *Number the b batches in a partial schedule σ from 1 to b such that*, $C^{[k]}(\sigma) \leq C^{[k-1]}(\sigma)$, $k \in \{1, 2, \ldots, b-1\}$. *If* $j \in j^{[k]}(\sigma)$, $r_j \leq R^{[k-1]}(\sigma)$ *and* $|J^{[k-1]}(\sigma)| < B$, *then TWT will not be increased by moving job j to batch* $k-1$ *from batch k*.

Proposition 2: *Consider two batches* B_a *and* B_b *of the same family such that* B_a *and* B_b *are scheduled with no other batches from the same family between them*, B_a *is scheduled before* B_b, B_a *has fewer than B jobs*, *and the ready time of* B_a *is greater than or equal to the ready time of job j in* B_b. *TWT will never increase as a result of moving job j to* B_a *from* B_b.

Proof: This is a straight-forward application of K Theorem 1 where the batches of the different families are considered separately.

The following example will highlight how Proposition 2 can be used to reduce the number of possible child nodes to consider. Consider 2 jobs in the same family with a processing time of 8. Job 1 is in batch A and is ready at time 6. Job 2 is in batch *B* and is ready at time 2. The maximum batch size is 2. If batch A is scheduled before batch *B*, then the conditions of Proposition 2 hold. Before Proposition 2 is applied, job 1 completes at time 14 and job 2 completes at time 22, as shown in figure 1*a*. After Proposition 2 is applied, job 1 still completes at time 14 and now, so does job 2, as shown in figure 1*b*. In this situation, if $\mathbf{U} = \{1, 2\}$, the child node of 1 and 2 batched together dominates the child node of job 1 placed in a batch by itself.

Proposition 2 is used to create Dominance Property 2. We present Dominance Property 2 for a single family with n unscheduled jobs, noting that it is extended to multiple incompatible families by applying it separately for each family. In the first step, we create all the nodes with the first available job and from 0 to $B-1$ additional members. In the second step, we create nodes with *B* total members, selected from the jobs not including the first available job. The key to reducing the number of nodes created is that we only create nodes that have less than *B* elements if they contain the earliest available job.

Dominance Property 2: Consider a node representing a partial sequence σ, let **U** be the set of jobs not in σ where $|\mathbf{U}| = n$ and all jobs are from the same family. Sort the jobs in **U** in increasing order of ready times. Let [*j*] be the *j*th job in the sorted list. We only need to consider child nodes created by the following algorithm:

1. For $k = 0$ to $\max(B-1, n-1)$
Create $\binom{n-1}{k}$ nodes with job [1] and $k-1$ of the remaining $n-1$ nodes.
2. For $k = 2$ to $n-B+1$
Create $\binom{n-k}{B-1}$ nodes with job [*k*] and $B-1$ of the last $n-k$ nodes in the sorted list.

To better understand the given algorithm, consider that the first step completely replicates the method to create the child nodes when Dominance Property 2 is not applied. Step 2 results when dominated nodes are not generated; for example, if any node is generated whose constituent jobs are a subset of the jobs in an existing node but do not contain the first job (which is ready earliest), application of Proposition 2 tells us that

Table 1. Counter-example problem data.

Job j	w_j	d_j	p_j	d_j/w_j
1	1	5	20	5
2	40	20	20	0.5
3	3	21	20	7
4	5	35	20	7

it is dominated by the existing node. The result is that only full batches will be generated in Step 2. If we have $n \geq B$ unscheduled jobs in the same family, we now have

$$\sum_{k=0}^{B-1} \binom{n-1}{k} + \sum_{k=2}^{n-B+1} \binom{n-k}{B-1}$$

possible child nodes. Before applying Dominance Property 2, when $n \geq B$, we had $\sum_{l=1}^{B} \binom{n}{l}$ possible child nodes. If $B=4$ and $n=6$, this reduces the number of child nodes from 56 to 31.

It would be convenient if a simple optimal ordering existed for the TWT static batch processing machine scheduling problem with incompatible job families. Using the weighted earliest due date ordering appears attractive, based on MU Proposition 3 (Proposition 3 in Mehta and Uzsoy (1998) shown below) and U Proposition 2, described earlier. MU Proposition 3 applies to $1|p-batch, incompat|\Sigma T_i$ and assumes that jobs in the same family are indexed in non-decreasing order of due dates.

MU Proposition 3: *There exists an optimal schedule* [*for* $1|p-batch, incompat|\Sigma T_j$ *where jobs in the same family are indexed in non-decreasing order of due dates*] *where all batches of the same family contain consecutively indexed jobs. Furthermore, for all batches of the same family j, the latest due date in one batch is no larger than the earliest due date of the next batch.*

The following counter-example shows that this is not the case: the weighted earliest due date order may not lead to an optimal schedule for $1|p-batch|\Sigma w_j T_j$.

Counter-example: Consider four jobs in the same family with a maximum batch size of two and other relevant data as shown in table 1. The weighted earliest due date orderings (using increasing order of d_j/w_j) are 2–1–3–4 or 2–1–4–3. The schedule (2, 1), (3, 4) yields a TWT of 97 while the schedule (2, 3), (1, 4) yields a TWT of 60.

3.5 *Branching decisions*

When determining which child node to consider next, we consider the most recently created node. This is due, in part, to our desire to utilise a stack data structure, in which the most recently created node is first accessed (LIFO). Other rules to determine which node should be evaluated first, such as 'highest LB first'or 'first come first served'have not been evaluated.

Table 2. Example problem data.

Job j	w_j	r_j	d_j	p_j	Family
4	8	0	15	4	1
7	4	1	18	10	2
8	2	3	22	10	2
3	9	4	18	4	1
6	3	6	24	10	2
1	8	7	16	4	1
2	5	9	19	4	1
5	2	10	25	10	2

3.6 *Example problem*

Table 2 shows eight jobs belonging to two families, with weights, ready times, due dates and processing times as given. Jobs 1–4 belong to family 1 and jobs 5–8 belong to family 2. The data has already been sorted by non-decreasing ready times. The maximum batch size is $B=2$. For this problem data, UB from BIA (Kurz and Mason 2005) is 99, which is shown in figure 2 as the root node of the tree. Figure 2 illustrates the nodes described in the following, with each node containing the jobs in the most recently added batch as well as the lower bound computed for that node and an indication of whether that node is fathomed. In the execution of the example, note that we do not expand nodes in the same order as is implemented.

In the initial stage, $\mathbf{U}=\{1, 2, 3, 4, 5, 6, 7, 8\}$. The possible child nodes for the root node, after applying Dominance Property 2, are (4), (4, 3), (4, 1), (4, 2), (3, 1), (3, 2), (1, 2), (7), (7, 8), (7, 6), (7, 5), (8, 6), (8, 5), (6, 5). We shall refer to these child nodes as Level 1 nodes, as indicated in figure 2.

Consider job 4 by itself in a node at Level 1. Job 4 starts at 0 and completes at 4; CT, the completion time of the last batch on the machine, is 4 as well. Since job 4 is not due until 15, it will never be tardy no matter how the following batches are constructed and $\mathrm{LB}(\mathbf{S})=0$. In order to calculate $\mathrm{LB}(\mathbf{U})$ for this partial schedule, we replace the ready times of the unscheduled jobs by the modified ready times. The computation for this node's $\mathrm{LB}(\mathbf{U})$ is shown in table 3. LB is calculated by aggregating $\mathrm{LB}(\mathbf{S})$ and $\mathrm{LB}(\mathbf{U})$, so $\mathrm{LB}=0$. Table 4 shows LBs for all Level 1 nodes as well as whether each is fathomed or not, based on the relation between each level 1 node's LB and UB. This information is also shown in figure 2. All Level 1 nodes except (7, 5) have

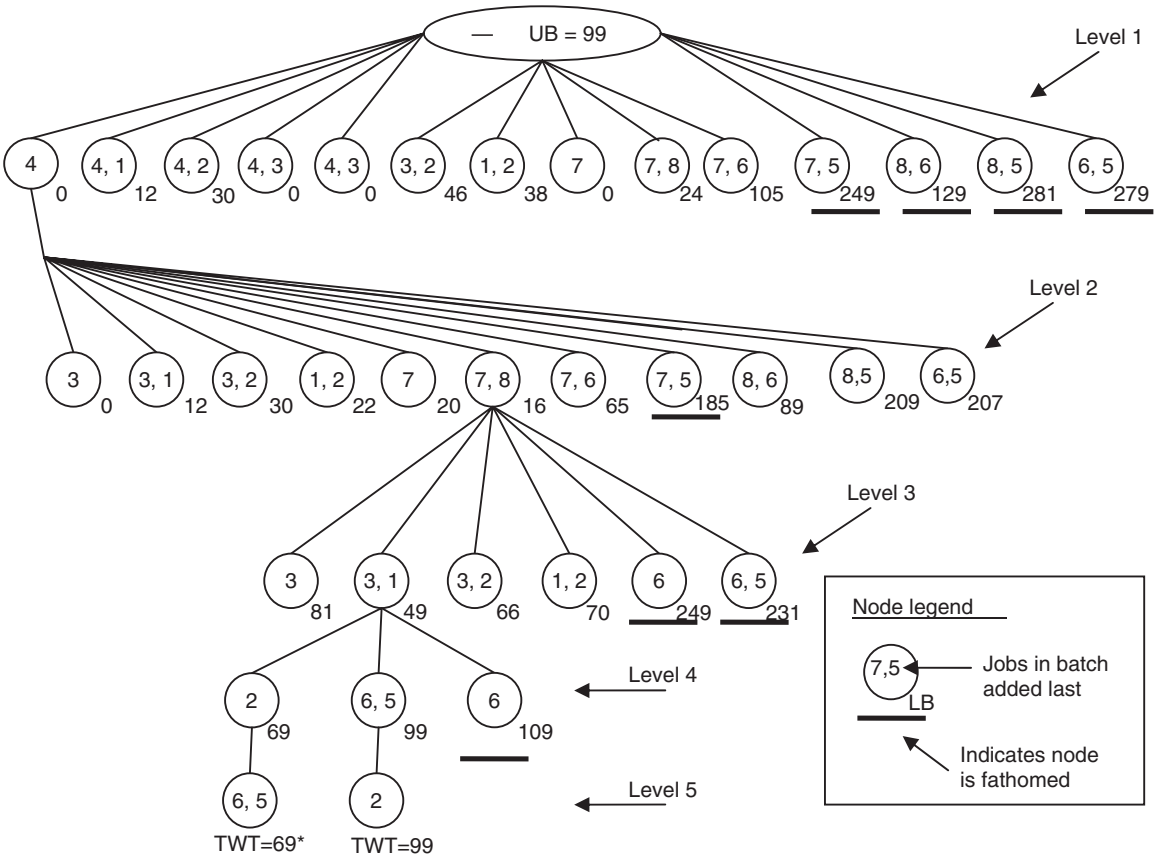

Figure 2. Nodes for the example problem.

Table 3. First node's LB(**U**) calculation.

Job j	w_j	r_j'	d_j	p_j	Best completion time	Best tardiness	Best weighted tardiness
7	4	4	18	10	14	0	0
8	2	4	22	10	14	0	0
3	9	4	18	4	8	0	0
6	3	6	24	10	16	0	0
1	8	7	16	4	11	0	0
2	5	9	19	4	13	0	0
5	2	10	25	10	20	0	0

LB(**U**) = 0

Table 4. Lower bounds for level 1 nodes.

S	LB(S)	LB(**U**)	LB	Fathomed because LB > Current UB?
4	0	0	0	N
(4, 1)	0	12	12	N
(4, 2)	0	30	30	N
(4, 3)	0	0	0	N
(3, 1)	0	12	12	N
(3, 2)	0	46	46	N
(1, 2)	0	38	38	N
(7)	0	0	0	N
(7, 8)	0	24	24	N
(7, 6)	0	105	105	Y
(7, 5)	8	249	257	Y
(8, 6)	0	129	129	Y
(8, 5)	0	281	281	Y
(6, 5)	0	279	279	Y

LB(**S**) values of zero. The non-zero LB(**S**) value for (7, 5) is found as follows. The batch (7, 5) can start at time 10, since that is the later of the ready times of the two jobs. It completes at time 20, as family 2's processing time is 10. Job 5 is not tardy, but job 7 is tardy by 2 time units. Since the weight on job 7 is 4, the resulting TWT of the partial schedule represented by node (7, 5) is 8.

Assume that the node selected to expand is node 4. The possible Level 2 child nodes are (7), (7, 8), (7, 6), (7, 5), (8, 6), (8, 5), (6, 5), (3), (3, 1), (3, 2), (1, 2), after applying Dominance Property 2. The lower bounds and results of the fathoming decisions are shown in Level 2 of figure 2. Level 3 of figure 2 is found if jobs (7, 8) are selected as the second batch after job 4. The second

Table 5. Complete schedule for new incumbent.

Batch no.	Job j	w_j	r_j	d_j	p_j	Family	C_j	TWT
1	4	8	0	15	4	1	4	0
2	7	4	1	18	10	2	14	0
	8	2	3	22				0
3	3	9	4	18	4	1	18	0
	1	8	7	16				16
4	2	5	9	19	4	1	22	15
5	6	3	6	24	10	2	32	24
	5	2	10	25				14
								69

Table 6. Complete schedule for optimal schedule.

Batch no.	Job j	w_j	r_j	d_j	p_j	Family	C_j	TWT
1	4	8	0	15	4	1	8	0
	3	9	4	18				0
2	1	8	7	16	4	1	13	0
	2	5	9	19				0
3	7	4	1	18	10	2	23	20
	6	3	6	24				0
4	8	2	3	22	10	2	33	22
	5	2	10	25				16
								58

batch completes at time 14. If jobs (3, 1) are selected as the third batch, we have $\mathbf{S} = \{4, 7, 8, 1, 3\}$ and the third batch ends at 18. The possible Level 4 child nodes of node (3, 1) after application of Dominance Property 2 are (2), (6) and (6, 5).

The only remaining batches to consider are (2) and (6, 5); the schedules are completely determined by selecting either batch first, as the other batch will form the only child node of the selected batch. Therefore, the LB in Level 4 of figure 2 contains the actual TWT for each unfathomed option. We see that the batches (4), (7, 8), (3, 1), (2), (6, 5) result in a TWT of 69, lower than the current upper bound. Therefore, UB = 69 and (4), (7, 8), (3, 1), (2), (6, 5) is the new incumbent solution, marked with an asterisk. The schedule is shown in table 5.

An optimal schedule can be obtained by following the same steps as described above until there is no node for expansion. Table 6 gives an optimal schedule for this problem with a TWT of 58.

4. Experimental investigation

We pattern our experimental design after the one presented by Mehta and Uzsoy (1998) to test the proposed B&B algorithm. We considered three classes of problems characterised by six factors: the number of jobs per family, the number of families, the job processing times, the maximum batch size, the range of job ready times and the range of job due dates. All jobs have integer weights drawn uniformly from 1 to 10. The value of each family's processing time in each class of problem was drawn from the same discrete uniform distribution shown in table 7. Moreover, the levels for the range of jobs ready times and the range of job due dates were identical for all problem classes and are shown in table 7 as well. The ready time factor has two levels corresponding to the value of α. When α is low, the ready times are closer to zero and therefore have a smaller range. When α is high, the ready times may be further from zero and therefore have a larger range. The due date factor has two values which are based on the values of two parameters, R and T. T is the expected percentage of tardy jobs and R is a range parameter. When both take on their low values, due dates are closer together. When both take on their high values, due dates have a larger range. In both cases, due dates are centred on a function of the estimated makespan. The levels of the factors that depend on problem class are shown in table 7 as well.

In total, each of the three classes contains $1 \times 2 \times 2 \times 1 \times 2 \times 2 = 16$ design points. Class I represents very small problems with 8 to 12 total jobs, Class II represents small problems with 15 to 20 total jobs and Class III represents medium problems with 24 to 32 total jobs. The characteristics of Class II and Class III differ only in the number of jobs per family. As 10 replicates are generated for each design point, a total of 480 problem instances are generated and examined. For each instance, the resulting schedule's TWT, computation time and the number of explored nodes are recorded.

5. Results and discussion

The developed algorithm is implemented in C++ and tested on a SunFire v480 with 2 900 MHz CPUs and 8 GB RAM. All test problems are allowed to run up to 3600 CPU seconds and all problems in Classes I and II were solved in this time limit. However, due to memory limitations, several problems failed to run longer than 2700 CPU seconds in Class III. The distribution of unsolved problems is shown in table 8. All 30 unsolved problems have 32 jobs.

Each line in tables 9 and 10 represents 40 test problems. The complete proposed B&B algorithm, with both dominance properties applied and BIA as the upper bound at the root node, was run on each problem instance, and table 9 summarises the overall results by problem size, as well as by level of the ready

Table 7. Problem characteristics common to all problem classes.

Problem factor		Values used	Total values per class
Jobs per family n_j	Class I Class II Class III	4 5 8	1
Number of families f	Class I Class II Class III	2, 3 3, 4 3, 4	2
Maximum batch size B	Class I Class II Class III	2, 3 4, 8 4, 8	2
Family processing time p_f	$P(p_f=2)=P(p_f=4)=P(p_f=16)=0.2$ $P(p_f=10)=0.3$ $P(p_f=20)=0.1$		1
Job ready time r_j	$\alpha \in \{0.5,\ 1.5\}$ $r_j \in [0, \alpha \cdot C\ \max]$, where $C\max = \frac{n_j \cdot f \cdot E[p_f]}{B}$		2
Job due date d_j	$R=0.5$ and $T=0.3$ Or $R=2.5$ and $T=0.6$ $r_j \in \left[\left\lceil \mu - \frac{\mu R}{2} \right\rceil, \mu + \frac{\mu R}{2}\right]$, where $\mu = C\max(1-T)$		2

Table 8. Distribution of unsolved problems.

		Ready time variability (*, *, *, *, α, *)		Due date tightness (*, *, *, *, *, RT)		Maximum batch size (*, *, B, *, *, *)	
Problem Class	(n_j, f, *, *, *, *)	0.5	1.5	$R=0.5$ $T=0.6$	$R=2.5$ $T=0.6$	4	8
III	(8, 4, *, *, *, *)	20	10	18	12	28	2

Note: There are 40 problem instances in each design point.

time and the due dates. The average computation times of various instances in CPU seconds and the average number of explored nodes are shown in table 9.

5.1 *Impact of total number of jobs*

From table 9, it is evident that as the total number of jobs increases, the average computation time as well as the number of explored nodes increases. This is consistent with most B&B results. The computation time and number of explored nodes are fairly small for 15 or fewer total jobs. The computation time increases by two orders of magnitude when moving from 15 to 20 jobs, and by one to two orders of magnitude when moving from 20 to 24 jobs. The number of explored nodes generally increases by one order of magnitude with the same increases in total number of jobs. When the total number of jobs increases to 32 from 24, we see the limits of the proposed B&B algorithm are reached. Not only do nearly half of these problems fail to solve in the time allowed, but the number of nodes explored for the solved problems are again higher by an order of magnitude. When the total number of jobs increases to 32 from 24, the computation time on average does not increase when ready times and due dates are drawn from a smaller range. Dominance Property 2 reduces the number of child nodes created in these cases and Dominance Property 1 allows us to find optimal schedule completions more often in these cases.

5.2 *Impact of ready times*

As the range of ready times increases, computation time and number of nodes decrease as per our expectations. When the range of ready times is high, a relatively larger number of dominant nodes are fathomed in the initial stages of the algorithm as compared to the smaller range of ready times, due to the application of Dominance Property 2. This property reduces the possible number of explored nodes, hence less computation time.

5.3 *Impact of due dates*

In most cases, computation time is less when due dates are centred at an earlier date. One of the possible explanations is the impact of Dominance Property 1. When due dates are centred at an earlier date, after a certain stage all jobs are due. In some cases, after a certain point, all jobs are ready as well. At this point, we can apply Dominance Property 1 and it is easy to find an optimal schedule of the remaining jobs. This property reduces the possible number of explored nodes, hence less computation time.

5.4 *Impact of maximum batch size*

Table 10 contains the average CPU time in seconds and average number of nodes explored, aggregated by the number of families and the maximum batch size. It is interesting to note that as the maximum batch size increases, computation time as well as the number of explored nodes decrease when other parameters are held constant. This is directly attributable to the impact of Dominance Property 2.

6. Conclusions and extensions

This research deals with the problem of minimising TWT on a single batch processing machine with job ready times. A dominance property has been developed and used in calculating the lower bound of nodes in the B&B tree. A different dominance property has been used to fathom nodes quickly based on the ready times of the jobs in the batch. Using these results we constructed a branch-and bound algorithm to solve problems involving up to 32 jobs.

Results show that as the batch size increases the number of explored nodes decrease and also as the ready time range increases the computation time decreases. Also, increasing the problem size increases computation time and this is more significant as the number of jobs exceeds 20.

Table 9. Average characteristics of solved problems.

		Ready time variability (*, *, *, *, α, *)				Due date tightness (*, *, *, *, *, RT)					
		0.5		1.5		$R=0.5$ $T=0.3$		$R=2.5$ $T=0.6$		Overall average	
Problem class	(n_j, f, *, *, *, *)	CPU time (s)	Number nodes explored	CPU time (s)	Number nodes explored	CPU time (s)	Number nodes explored	CPU time (s)	Number nodes explored	CPU time (s)	Number nodes explored
I	(4, 2, *, *, *, *)	<0.01	50	<0.01	27	<0.01	37	<0.01	40	<0.01	39
	(4, 3, *, *, *, *)	0.09	309	0.02	63	0.07	227	0.04	144	0.05	186
II	(5, 3, *, *, *, *)	0.08	331	0.11	274	0.10	300	0.09	305	0.10	303
	(5, 4, *, *, *, *)	3.14	7052	3.02	5344	2.867	5478	3.29	6918	3.08	6198
III	(8, 3, *, *, *, *)	177.22	72 556	23.90	8963	268.27	60 493	66.99	21 029	100.57	40760
	(8, 4, *, *, *, *)	141.88	285 514	457.62	402 856	261.52	389 084	491.49	327 027	331.33	328 605

Note: Class III problems with 4 families only include values for solved problems.
The * indicates aggregation across all levels of the factor.

Table 10. Impact of batch size on computations in class I problems.

		Maximum batch size (*, *, *, *, B)			
		Low level 2 for class I 4 for class II and III		High level 3 for class I 8 for class II and III	
Problem class	(n_j, f, *, *, *, *)	Average CPU time (s)	Average nodes explored	Average CPU time (s)	Average nodes explored
I	(4, 2, *, *, *, *)	<0.01	52	<0.010	25
	(4, 3, *, *, *, *)	0.08	259	0.029	112
II	(5, 3, *, *, *, *)	0.13	389	0.060	216
	(5, 4, *, *, *, *)	4.64	8074	1.520	4321
III	(8, 3, *, *, *, *)	182.24	77540	18.880	3979
	(8, 4, *, *, *, *)	569.46	63 1299	304.100	22 8478

Note: Class III problems with 4 families only include values for solved problems.
The * indicates aggregation across all levels of the factor.

In order to improve the algorithm for larger size problems, we would need to develop additional dominance properties. However, we feel a more fruitful use of this work is as an improvement method in a parallel machine version of the problem, where we can use the B&B in a truncated form for each machine separately. Some other potential extensions of this work involve sequence dependent set up times and compatible job families. Clearly, a B&B algorithm such as this one can be used by researchers developing heuristic and meta-heuristic solution methods for $1|p-batch, r_j, incompat|\Sigma w_j T_j$ to evaluate the effectiveness of their algorithms on small problems.

References

Azizoglu, M. and Webster, S., Scheduling a batch processing machine with non-identical job sises. *Int. J. Prod. Res.*, 2000, **38**, 2173–2184.

Azizoglu, M. and Webster, S., Scheduling a batch processing machine with incompatible job families. *Comp. Indust. Eng.*, 2001, **39**, 325–335.

Balasubramanian, H., Monch, L., Fowler, J. and Pfund, M., Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimise total weighted tardiness. *Int. J. Prod. Res.*, 2004, **42**, 1621–1638.

Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M.Y., Potts, C.N., Tautenhahn, T. and van de Velde, S.L., Scheduling a batching machine. *J. Sched.*, 1998, **1**, 31–54.

Brucker, P. and Knust, K., 2006, Complexity results for scheduling problems. Available online at: http://www.mathematik.uni-osnabrueck.de/research/OR/class/ (accessed 20 February 2006).

Chandru, V., Lee, C.Y. and Uzsoy, R., Minimising total completion time on batch processing machine. *Int. J. Prod. Res.*, 1993, **31**, 2097–2121.

Cheraghi, S.H., Vishwaram, V. and Krishnan, K.K., Scheduling a single batch-processing machine with disagreeable ready times and due dates. *Int. J. Indust. Eng.*, 2003, **10**, 175–187.

Du, J. and Leung, J.Y.-T., Minimising total tardiness on one processor is NP-hard. *Math. Oper. Res.*, 1990, **3**, 483–495.

Dupont, L. and Dhaenens-Flipo, C., Minimising the make-span on a batch machine with non-identical job sizes: an exact procedure. *Comp. Oper. Res.*, 2002, **29**, 807–819.

Dupont, L. and Ghazvini, F.J., A branch and bound algorithm for minimising mean flow time on a single batch processing machine. *Int. J. Indust. Eng.*, 1997, **4**, 197–203.

Graham, R.L., Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G., Optimisation and approximation in deterministic sequencing and scheduling theory: a survey. *Ann. Disc. Math.*, 1979, **5**, 287–326.

Hochbaum, D.S. and Landy, D., Scheduling semiconductor burn-in operations to minimise total flowtime. *Oper. Res.*, 1997, **45**, 874–885.

Kurz, M.E. and Mason, S. J., Minimising total weighted tardiness on a batch-processing machine with incompatible job families and job ready times. *Int. J. Prod. Res.*, 2005, in press (submitted in July 2005).

Kurz, M.E., On the structure of optimal schedules for minimising total weighted tardiness on parallel, batch-processing machines, in *Proceedings of the Industrial Engineering Research Conference*, Portland, OR (CD-ROM), 2003.

Li, C.-L. and Lee, C.-Y., Scheduling with agreeable release times and due dates on a batch processing machine. *Euro. J. Op. Res.*, 1997, **96**, 564–569.

Mathirajan, M. and Sivakumar, A.I., Scheduling of batch processors in semiconductor manufacturing. A review, in *Proceedings of the Singapore MIT Alliance*, 2003. Available online at: https://dspace.mit.edu/retrieve/3521/IMST021 (accessed 20 February 2006).

Mehta, S.V. and Uzsoy, R., Minimising total tardiness on a batch-processing machine with incompatible job families. *IIE Trans.*, 1998, **30**, 165–178.

Monch, L., Balasubramanian, H., Fowler, J.W. and Pfund, M.E., Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Comp. Oper. Res.*, 2005, **32**, 2731–2750.

Perez, I., Fowler, J.W. and Carlyle, W.M., Minimising total weighted tardiness on a single batch processing machine with incompatible job families. *Comp. Oper. Res.*, 2005, **32**, 327–341.

Potts, C.N. and Kovalyov, M.Y., Scheduling with batching: a review. *Euro. J. Op. Res.*, 2000, **120**, 228–249.

Qi, X. and Tu, F., Earliness and tardiness scheduling problems on a batch processor. *Disc. Appl. Math.*, 1999, **98**, 131–145.

Rinnooy Kan, A.H.G., *Machine Scheduling Problems*, 1976 (Martinus Nijhoff: The Hague, The Netherlands).

Sung, C.S. and Choung, Y.I., Minimising makespan on a single burn-in oven in semiconductor manufacturing. *Euro. J. Oper. Res.*, 2000, **120**, 550–574.

Sung, C.S., Choung, Y.I., Hong, J.M. and Kim, Y.H., Minimising makespan on a single burn-in oven with job families and dynamic job arrivals. *Comp. Oper. Res.*, 2002, **29**, 995–1007.

Uzsoy, R. and Yang, Y., Minimising total weighted completion time on a single batch processing machine. *Prod. Op. Manage.*, 1997, **6**, 57–73.

Uzsoy, R., Scheduling a single batch processing machines with non identical job sizes. *Int. J. Prod. Res.*, 1994, **32**, 1615–1635.

Uzsoy, R., Scheduling batch processing machines with incompatible job families. *Int. J. Prod. Res.*, 1995, **33**, 2685–2708.

Wang, C.S. and Uzsoy, R., A genetic algorithm to minimise maximum lateness on a batch processing machine. *Comp. Oper. Res.*, 2002, **29**, 1621–1640.

Webster, S. and Baker, K.R., Scheduling groups of jobs on a single machine. *Oper. Res.*, 1995, **43**, 692–703.

Dr. Mary E. Kurz is Assistant Professor of Industrial Engineering at Clemson University. She earned her MS in Systems Engineering and PhD in Systems and Industrial Engineering from the University of Arizona. Her research interests include scheduling and genetic algorithms, for both single and multiple objective problems.

Sarath Tangudu holds a MS degree in Industrial Engineering from the Clemson University. His research interests include developing algorithms and heuristics for production scheduling problems in semi conductor industry. He has worked as an Operations Research Analyst at Southern Company, Birmingham for the past 2 years. He has been involved with simulation and optimisation in resource planning and supply chain.