# BRANCH AND BOUND ALGORITHMS FOR SINGLE MACHINE SCHEDULING WITH BATCH SET-UP TIMES TO MINIMIZE TOTAL WEIGHTED COMPLETION TIME

by

H.A.J. CRAUWELS*
A.M.A HARIRI**
C.N. POTTS†
and
L.N. VAN WASSENHOVE††

95/70/TM

* Professor at KIHDN, J. De Nayerlaan 5, 2860 Sint-Katelijine-Waver, Belgium.

** Professor at King Abdul-Aziz University, Jeddah 21413, Saudi Arabia.

† Professor at University of Southampton, Southampton SO17 1BJ, UK.

†† Professor at INSEAD, Boulevard de Constance, Fontainebleau 77305 Cedex, France.

# Branch and Bound Algorithms for Single Machine Scheduling with Batch Set-Up Times to Minimize Total Weighted Completion Time

H. A. J. Crauwels
*KIHDN*
*J. De Nayerlaan 5*
*2860 Sint-Katelijne-Waver, Belgium*

A. M. A. Hariri
*Department of Statistics*
*King Abdul-Aziz University*
*Jeddah 21413, Saudi Arabia*

C. N. Potts
*Faculty of Mathematical Studies*
*University of Southampton*
*Southampton SO17 1BJ, United Kingdom*

L. N. Van Wassenhove
*INSEAD*
*Boulevard de Constance*
*77305 Fontainebleau, France*

July, 1995

**Abstract**

This paper presents several branch and bound algorithms for a single machine scheduling problem with batching. Jobs are partitioned into families, and a set-up time is necessary when there is a switch from processing jobs of one family to jobs of another family. The objective is to minimize the total weighted completion time. A lower bound based on Lagrangean relaxation of the machine capacity constraint is derived. Also, a multiplier adjustment method to find values of the multipliers is proposed. Computational experience with instances having up to 70 jobs shows that the lower bounds are effective in restricting the search.

## 1 Introduction

Many practical scheduling problems involve processing several families of related jobs on common facilities, where a set-up time is incurred whenever there is a switch from processing a job in one family to a job in another family. To avoid set-ups, several jobs of the same family may be scheduled contiguously to form a batch. A schedule defines how batches are formed, and specifies the processing order of the batches.

As an example of a problem that involves scheduling and batching, Ahn and Hyun [1] describe an application in the manufacture of steel pipes. The manufacturing process for pipes requires the use of a rolling machine, where the roller to be used depends on the outer diameter of the pipe. A change in the roller corresponds to a set-up. Thus, orders for pipes with the same outer diameter lie within the same job family in our model. A review of algorithms and complexity for various scheduling problems that involve batching is given by Potts and Van Wassenhove [14].

In this paper, we consider a single machine scheduling problem with $N$ jobs which are partitioned into $F$ families. Set-up times are sequence independent: each depends only on the family of the job to be processed next. We assume that all the jobs are available at time zero and that each job has a given processing time and an associated positive weight. We wish to find a schedule which minimizes the total weighted completion time of the jobs.

When all set-up times are zero, the problem is solved in $O(N \log N)$ time by sequencing the jobs in shortest weighted processing time (SWPT) order (non-decreasing order of processing time to weight ratios). Monma and Potts [11] show that, for non-zero set-up times, jobs within each family are sequenced in SWPT order in an optimal schedule. Using this property, they derive a dynamic programming algorithm which requires $O(F^2N^{2F})$ time. More efficient dynamic programming algorithms have been developed subsequently. For the case of unit weights, Ahn and Hyun [1] develop an algorithm which requires $O(F^2N^F)$ time, and Ghosh [5] proposes an $O(F^2N^F)$ algorithm for arbitrary weights. Thus, for fixed $F$, the problem is polynomially solvable, although the algorithms provide a practical method of solution only when $F$ is very small. The issue of whether the problem is $NP$-hard for arbitrary $F$ is unresolved, even for the case of unit weights.

Heuristic methods are also the subject of some studies. For the case of unit weights, Gupta [6] devizes a SPT-based heuristic. Essentially, it is a single-pass greedy procedure that requires $O(N \log N)$ time: the job to be processed next is chosen so that it has the smallest completion time amongst all candidate jobs. Although having the advantage of modest computational requirements, the quality of the schedules that it generates is often rather poor.

Various local search heuristics are also proposed in the literature. Ahn and Hyun [1] develop a descent (or iterative improvement) heuristic. They use a neighbourhood in which blocks of jobs of the same batch are shifted from one part of the sequence to another, but with the restriction that the SWPT ordering of jobs within a family is maintained. Mason [9] proposes a genetic algorithm which uses the observation that knowledge of which job starts each batch enables a solution to be constructed by ordering the batches using a generalization of the SWPT rule. Thus, solutions can be represented as binary strings to which standard genetic operators can be applied. Crauwels et al. [3] develop simulated annealing, threshold accepting and tabu search methods which build on the work of Ahn and Hyun. In computational tests, Crauwels et al. find their tabu search algorithm to be superior to a multi-start version of the descent method of Ahn and Hyun, to simulated annealing and threshold accepting, and also to Mason's genetic algorithm.

Mason and Anderson [10] propose a branch and bound algorithm. A special feature of their algorithm is the extensive use of dominance rules to restrict the size of the branch and bound search tree. They use a forward branching rule which yields partial schedules in which jobs are sequenced in the initial positions. Their lower bound is derived using objective splitting: the total weighted completion time can be partitioned into contributions from the processing times and from the set-up times, which are optimized separately. The SWPT rule minimizes the first contribution. A generalized SWPT rule applied to families, where the processing time of a family is replaced by its set-up time, minimizes the second. Computational results indicate that this algorithm represents a practical method of solution provided that there are no more than about 30 jobs. The restriction to fairly small instances is attributed to the weakness of the lower bound: the dominance rules are the major contributors to the pruning of the branch and bound search tree.

In spite of significant recent research activity in the area of scheduling models which involve batching, there have been few attempts to design computationally effective enumerative algorithms. This is unfortunate in view of the practical importance of these problems. In this paper, we propose a new branch and bound algorithm which is superior to that of Mason and Anderson [10]. It uses lower bounds that are derived from a Lagrangean relaxation of machine capacity constraints. A special feature is the use of a multiplier adjustment method for obtaining values of the multipliers. The ability to construct multipliers with some desirable properties reinforces the view that it is often possible to exploit problem structure in a Lagrangean procedure, rather than resort to the heavy-handed subgradient optimization approach [2,7,12,13]. Our computational results demonstrate that the quality of the lower bounds obtained from the multiplier adjustment procedure is high.

In Section 2, we give a formal statement of our problem and present some dominance criteria. Section 3 derives our lower bound using a Lagrangean relaxation of the machine capacity constraints. Our multiplier adjustment procedure for finding values of the multipliers is also presented. Section 4 gives a heuristic method for scheduling the jobs. Branch and bound algorithms are described in Section 5. Section 6 reports on computational experience with the branch and bound algorithms, and some concluding remarks are given in Section 7.

## 2 Problem structure and dominance rules

To state our problem of scheduling with batch set-up times more precisely, we are given a set $\mathcal{N}$ containing $N$ jobs that are divided into $F$ families. Each family $f$, for $f = 1, \ldots, F$, contains $n_f$ jobs. All jobs become available for processing at time zero and are to be scheduled on a single machine. For the $i$'th job of family $f$, which we denote by $(i, f)$, let $p_{if}$ denote its processing time and $w_{if}$ its positive weight. We assume that the jobs within each family are indexed in SWPT order. Thus, $p_{1f}/w_{1f} \leq \cdots \leq p_{n_f,f}/w_{n_f,f}$ for $f = 1, \ldots, F$.

A sequence independent set-up time $s_f \geq 0$ is incurred whenever a job in family $f$ is processed immediately after a job in a different family. Also, an initial set-up time $s_f$ is required if a job from family $f$ is processed first on the machine. Thus, the processing of each batch must be preceded by the relevant set-up.

We first present two fundamental results on the structure of an optimal schedule.

**Theorem 1** (Monma and Potts [11]). *In any optimal schedule, jobs within each family are sequenced in SWPT order.*

From Theorem 1, the problem reduces to one of merging SWPT-ordered lists of jobs for the different families to form a sequence.

Consider any schedule $S = (B_1, \dots, B_v)$, where $B_\nu$ ($\nu = 1, \dots, v$) is a batch comprising a maximal consecutive subsequence of jobs from the same family $f_\nu$. We define the weighted processing time (WPT) ratio for each batch $B_\nu$ to be

$$\mathrm{WPT}(B_\nu) = \left( s_{f_\nu} + \sum_{(i,f) \in B_\nu} p_{if} \right) \Big/ \sum_{(i,f) \in B_\nu} w_{if}. \tag{1}$$

The following result gives a generalized SWPT rule for batches that uses WPT ratios.

**Theorem 2** (Mason and Anderson [10]). *In any optimal schedule, batches are sequenced in nondecreasing order of WPT ratios.*

We now describe some dominance rules that allow us to restrict the set of candidates for sequencing next in the continuation of some initial partial schedule $S = (B_1, \dots, B_v)$, where $B_\nu$ ($\nu = 1, \dots, v$) is a batch. These rules are based on the results of Mason and Anderson [10]. We assume that $S$ satisfies the conditions of Theorem 1. Let $m_f$ denote the number of jobs of family $f$ ($f = 1, \dots, F$) that are contained in $S$, where $0 \le m_f \le n_f$.

Suppose that $B_v$ is a batch of some family $g$, where $g = f_v$, and that job $(m_h + 1, h)$, where $h \ne g$, is a candidate to be sequenced immediately after the last job of $S$. If $m_h \ne 0$, let job $(m_h, h)$ be contained in batch $B_u$, where $1 \le u < v$ and $h = f_u$.

Our dominance rules are as follows.

**Rule 1.** Suppose that $m_g < n_g$ and $\mathrm{WPT}(B_v) > p_{m_g+1,g}/w_{m_g+1,g}$. Then job $(m_g + 1, g)$ must be scheduled immediately after the last job of $S$.

**Rule 2.** Suppose that $v > 1$, and $\mathrm{WPT}(B_{v-1}) > \mathrm{WPT}(B_v)$. If $m_g = n_g$, then $S$ cannot form the start of an optimal schedule. If $m_g < n_g$, then job $(m_g + 1, g)$ must be scheduled immediately after the last job of $S$.

**Rule 3.** Suppose that $\mathrm{WPT}(B_v) > \min_{f \in \{f' | m_{f'} < n_{f'}\}} \{(s_f + \sum_{i=m_f+1}^{n_f} p_{if}) / \sum_{i=m_f+1}^{n_f} w_{if}\}$. If $m_g = n_g$, then $S$ cannot form the start of an optimal schedule. If $m_g < n_g$, then job $(m_g + 1, g)$ must be scheduled immediately after the last job of $S$.

**Rule 4.** Suppose that $m_g < n_g$ and $p_{m_h+1,h}/w_{m_h+1,h} > p_{m_g+1,g}/w_{m_g+1,g}$. Then there is no optimal schedule in which job $(m_h + 1, h)$ is scheduled immediately after the last job of $S$.

**Rule 5.** Suppose that $m_h > 0$ and either $p_{m_h,h}/w_{m_h,h} > \mathrm{WPT}'(B_{u+1}, \dots, B_v)$ or $\mathrm{WPT}'(B_{u+1}, \dots, B_v) > p_{m_h+1,h}/w_{m_h+1,h}$, where

$$\mathrm{WPT}'(B_{u+1}, \dots, B_v) = \left( s_h + \sum_{\nu=u+1}^{v} (s_{f_\nu} + \sum_{(i,f) \in B_\nu} p_{if}) \right) \Big/ \sum_{\nu=u+1}^{v} \sum_{(i,f) \in B_\nu} w_{if}.$$

Then there is no optimal schedule in which job $(m_h + 1, h)$ is scheduled immediately after the last job of $S$.

We now justify the validity of these rules. Rules 2 and 3 follow from Theorem 2. Under the conditions of Rule 2, schedule $S$ is not consistent with this WPT ordering unless further jobs are added to batch $B_v$. Rule 3 considers an upper bound on the smallest WPT value that can occur when batches of the unscheduled jobs are formed. If WPT($B_v$) exceeds this upper bound, further jobs must be added to batch $B_v$ in an attempt to satisfy the WPT ordering. The validity of Rules 1, 4 and 5 follow from Corollary 4.1, Corollary 4.2 and Theorem 4 of Mason and Anderson [10], respectively.

Rules 1, 2, 4 and 5 are each used by Mason and Anderson [10] in their branch and bound algorithm. However, Rule 3 is not used in any previous studies.

A *dynamic programming dominance rule* is also useful. For two initial partial schedules $S_1$ and $S_2$ which have identical final jobs and which contain the same jobs, if $S_2$ has a completion time of its last job and a total weighted completion time that are no smaller than the corresponding values for $S_1$, then $S_2$ is dominated by $S_1$.

Unfortunately, Rules 1 through 5 and the dynamic programming dominance rule can be used within the branch and bound search tree only when a forward branching rule is used. Attempts to make important decisions early through the use of a flexible branching rule have to be balanced against the loss of these dominance rules.

A useful *preprocessing* routine is suggested by Rule 1. Suppose that $n_f \geq 2$ and $(s_f + p_{1f})/w_{1f} > p_{2f}/w_{2f}$ for some family $f$. Rule 1 ensures that no optimal schedule contains a batch comprising only job $(1, f)$. Thus, jobs $(1, f)$ and $(2, f)$ must be scheduled contiguously. Using the theory of Lawler [8], they can be combined to form a single composite job with processing time $p_{1f} + p_{2f}$ and weight $w_{1f} + w_{2f}$. Although this composition increases the total weighted completion time by a constant term $p_{2f}w_{1f}$, an equivalent problem results. After reindexing the jobs, this preprocessing step is reapplied. Mason and Anderson [10] show that if $p_{i-1,f}/w_{i-1,f} = p_{if}/w_{if}$, where $(i, f) \in \mathcal{N}$ and $i > 1$, then there exists an optimal solution in which jobs $(i-1, f)$ and $(i, f)$ are scheduled contiguously. Thus, composite jobs are formed from such pairs of jobs.

## 3 Lower bounds

In this section, we derive a new lower bound based on Lagrangean relaxation. Two alternative methods of determining values of the multipliers using a multiplier adjustment technique and subgradient optimization are also described.

Based on ideas of Fisher [4], we give a time-indexed formulation of the problem. Let $T$ denote an upper bound on the completion time of the last job; for instance, $T = \sum_{(i,f)\in\mathcal{N}}(s_f + p_{if})$. Also, let $\mathcal{F} = \{1, \ldots, F\}$ and $\mathcal{T} = \{1, \ldots, T\}$. By defining variables

$$\begin{aligned}
C_{if} &= \text{ the completion time of job } (i,f), && (i,f) \in \mathcal{N}, \\
x_{ift} &= \begin{cases} 1 & \text{if job } (i,f) \text{ is processed in the interval } [t-1,t], \\ 0 & \text{otherwise,} \end{cases} && (i,f) \in \mathcal{N},\ t \in \mathcal{T}, \\
y_{ft} &= \begin{cases} 1 & \text{if a set-up for family } f \text{ occurs in } [t-1,t], \\ 0 & \text{otherwise,} \end{cases} && f \in \mathcal{F},\ t \in \mathcal{T},
\end{aligned}$$

we obtain the following formulation:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{(i,f)\in\mathcal{N}} w_{if} C_{if} & & \\
\text{subject to} \quad & \sum_{t\in\mathcal{T}} x_{ift} = p_{if} & (i,f)\in\mathcal{N} \quad & (2)\\
& \sum_{(i,f)\in\mathcal{N}} x_{ift} + \sum_{f\in\mathcal{F}} y_{ft} \le 1 & t\in\mathcal{T} \quad & (3)\\
& s_f(x_{ift} - x_{i,f,t-1} - x_{i-1,f,t-1}) \le \sum_{t'=t-s_f}^{t-1} y_{ft'} & (i,f)\in\mathcal{N},\ t = s_f+1,\ldots,T \quad & (4)\\
& x_{ift} = \begin{cases} 1 & \text{if } t\in\{C_{if}-p_{if}+1,\ldots,C_{if}\} \\ 0 & \text{otherwise} \end{cases} & (i,f)\in\mathcal{N} \quad & (5)\\
& C_{if} \ge C_{i-1,f} + p_{if} & (i,f)\in\mathcal{N} \quad & (6)\\
& C_{0f} \ge 0 & f\in\mathcal{F} \quad & (7)\\
& y_{ft} \in \{0,1\} & f\in\mathcal{F},\ t\in\mathcal{T}. \quad & (8)
\end{aligned}
$$

Constraints (2) ensure that the required processing of all jobs is executed, while the machine capacity constraints are represented by (3). Constraints (4) ensure that the necessary set-ups are performed: if job $(i,f)$ is processed in the interval $[t-1,t]$, but neither job $(i,f)$ nor job $(i-1,f)$ is processed in $[t-2,t-1]$, then job $(i,f)$ is the first job in a batch and starts its processing at time $t-1$. Consequently, a set-up for family $f$ is performed throughout the previous $s_f$ time intervals. Constraints (5) ensure that the variables $x_{ift}$ are consistent with the completion time variables $C_{if}$, and that no preemption is allowed. The SWPT ordering within families (Theorem 1) is represented by constraints (6).

To obtain a lower bound, we perform a Lagrangean relaxation of the machine capacity constraints (3). If $\mu = (\mu_1,\ldots,\mu_T)$ is a given vector of non-negative multipliers, then we obtain the lower bound

$$
L(\mu) = \min\Big\{ \sum_{(i,f)\in\mathcal{N}} w_{if}C_{if} + \sum_{t\in\mathcal{T}}\Big( \sum_{(i,f)\in\mathcal{N}} \mu_t x_{ift} + \sum_{f\in\mathcal{F}} \mu_t y_{ft}\Big)\Big\} - \sum_{t\in\mathcal{T}} \mu_t
$$

subject to (2), (4), (5), (6), (7) and (8).

We note that the Lagrangean problem decomposes; there is a separate subproblem for each family. In any subproblem, a cost $\mu_t$ is associated with performing processing or a set-up in the time interval $[t-1,t]$.

To solve the Lagrangean subproblem for family $f$ ($f = 1,\ldots,F$), we propose a dynamic programming algorithm. We define a recursion on $z(j,t)$ which represents the cost of scheduling jobs $(1,f),\ldots,(j,f)$, where job $(j,f)$ is completed at time $t$ and $t \ge s_f + \sum_{i=1}^{j} p_{if}$. All initial values of

$z(j,t)$ are set to $\infty$. We compute $z(j,t)$ for $t = s_f + \sum_{i=1}^{j} p_{if}, \ldots, T$ using the recursion

$$z(j,t) = \begin{cases} w_{1f}t + \displaystyle\sum_{t'=t-s_f-p_{1f}+1}^{t} \mu_{t'} & \text{for } j = 1, \\ \min\Big\{ z(j-1, t-p_{jf}) + w_{jf}t + \displaystyle\sum_{t'=t-p_{jf}+1}^{t} \mu_{t'}, & \\ \qquad \displaystyle\min_{t'=1,\ldots,t-p_{jf}-s_f} \{z(j-1,t')\} + w_{jf}t + \sum_{t'=t-s_f-p_{jf}+1}^{t} \mu_{t'} \Big\} & \text{for } j = 2, \ldots, n_f. \end{cases}$$

For $j = 1$, we note that $z(j,t)$ is the weighted completion time for job $(j,f)$ when it completes at time $t$ plus the sum of multipliers for the time periods during which $(j,f)$ is processed. However, for $j = 2, \ldots, n_f$, the contribution for jobs $(1,f), \ldots, (j-1,f)$ must be added: the two terms in the minimization consider the cases that jobs $(j-1,j)$ and $(j,f)$ are assigned to the same batch, and are scheduled in different batches, respectively. The minimum total cost for jobs of family $f$ is

$$K_f = \min_t \{z(n_f, t)\},$$

where the minimization is over values $t = s_f + \sum_{i=1}^{n_f} p_{if}, \ldots, T$. Thus, we obtain the lower bound

$$\mathrm{LB}(\mu) = \sum_{f \in \mathcal{F}} K_f - \sum_{t \in \mathcal{T}} \mu_t.$$

The computation of $\mathrm{LB}(\mu)$ requires $O(NT)$ time.

Ideally, we would like to choose $\mu$ so that $\mathrm{LB}(\mu)$ is as large as possible. We propose two alternative approaches for the determination of $\mu$. In the first, the *multiplier adjustment method* is used. This is a constructive procedure for computing multipliers from a heuristic schedule. First, we apply the heuristic of the next section to find a schedule $S$ for which the jobs within each family are sequenced in SWPT order. Let $S = (B_1, B_2, \ldots, B_v)$, where $B_\nu$ $(\nu = 1, \ldots, v)$ is a batch which completes at time $C(B_\nu)$. Also, let $T_0 = 0$, $T_\nu = C(B_\nu)$ for $\nu = 1, \ldots, v-1$ and $T_v = C(B_v) - 1$. After setting $\mu_1 = \sum_{(i,f)\in\mathcal{N}} w_{if} - 1/\mathrm{WPT}(B_v)$, where $\mathrm{WPT}(B_v)$ is computed from (1), other multipliers are computed recursively using

$$\mu_{t+1} = \begin{cases} \mu_t - 1/\mathrm{WPT}(B_\nu) & \text{for } \nu = 1, \ldots, v,\ t = T_{\nu-1}+1, \ldots, T_\nu, \\ \mu_t & \text{for } t = T_v + 1, \ldots, T-1. \end{cases} \tag{9}$$

This construction yields $\mu_t = 0$ for $t = T_v + 1, \ldots, T$. Note that, since $\{\mu_t\}$ is non-increasing in $t$, machine time is more expensive for earlier time periods than for later ones. This pricing structure is consistent with the observation that since more jobs are competing for machine time early in the schedule, such time periods should be assigned a higher price.

We now provide a theoretical justification for this choice of multipliers by showing that the lower bound is exact when every family contains a single job.

**Theorem 3** *If $S = (B_1, \ldots, B_F)$, where $B_f$ is a batch formed from a single job of family $f$ for $f = 1, \ldots, F$, then $\mathrm{LB}(\mu) = \sum_{f\in\mathcal{F}} w_{1f} C(B_f)$.*

**Proof.** When each family contains a single job, the heuristic method of the next section generates a schedule $S$ in which batches are ordered in non-decreasing order of WPT values. Thus, we assume without loss of generality that $S = (B_1, \ldots, B_F)$, where $\mathrm{WPT}(B_1) \leq \ldots \leq \mathrm{WPT}(B_F)$.

Let $c_{ft} = w_{1f}t + \sum_{t'=t-s_f-p_{1f}+1}^{t} \mu_{t'}$ denote the cost in the Lagrangean problem of scheduling job $(1, f)$ to complete at time $t$, for $f = 1, \ldots, F$ and $t = s_f + p_{1f}, \ldots, T$. We show first that $c_{ft} \leq c_{f,t+1}$ for $t = C(B_f), \ldots, T-1$. Clearly,

$$c_{f,t+1} - c_{ft} = w_{1f} - (\mu_{t-s_f-p_{1f}+1} - \mu_{t+1}). \tag{10}$$

From (9), we observe that $\mu_{t'+1} - \mu_{t'} = 0$ for $t' = C(B_F), \ldots, T-1$. Also, for $t' = C(B_f), \ldots, C(B_F) - 1$, we obtain from (9) that there is some family $f'$, where $f' \in \{f, \ldots, F\}$, for which $\mu_{t'} - \mu_{t'+1} = 1/\mathrm{WPT}(B_{f'})$: the indexing of batches using WPT values implies that $\mu_{t'} - \mu_{t'+1} \leq 1/\mathrm{WPT}(B_f)$. We have now established that $\mu_{t'} - \mu_{t'+1} \leq 1/\mathrm{WPT}(B_f)$ for $t' = C(B_f), \ldots, T-1$. Substituting in (10), we obtain

$$c_{f,t+1} - c_{ft} \geq w_{1f} - (p_{1f} + s_f)/\mathrm{WPT}(B_f). \tag{11}$$

From the definition of $\mathrm{WPT}(B_f)$ in (1), we obtain that the right-hand side of (11) is equal to zero, which establishes the desired inequality $c_{ft} \leq c_{f,t+1}$ for $t = C(B_f), \ldots, T-1$.

We now show that $c_{ft} \geq c_{f,t+1}$ for $t = s_f + p_{1f}, \ldots, C(B_f) - 1$. Analogously with our previous analysis, for $t' = s_f + p_{1f}, \ldots, C(B_f) - 1$, there is some family $f'$, where $f' \in \{1, \ldots, f\}$, for which $\mu_{t'} - \mu_{t'+1} = 1/\mathrm{WPT}(B_{f'}) \geq 1/\mathrm{WPT}(B_f)$. Therefore,

$$c_{f,t+1} - c_{ft} \leq w_{1f} - (p_{1f} + s_f)/\mathrm{WPT}(B_f) = 0,$$

which yields $c_{ft} \geq c_{f,t+1}$ for $t = s_f + p_{1f}, \ldots, C(B_f) - 1$, as required.

The above analysis shows that there exists an optimal solution of the Lagrangean problem in which each batch $B_f$ is scheduled to complete at time $C(B_f)$. Thus, each of the relaxed constraints (3) is satisfied with equality in a solution of the Lagrangean problem. This establishes that $\mathrm{LB}(\mu) = \sum_{f \in \mathcal{F}} w_{1f} C(B_f)$. $\square$

The tightness of the lower bound that we obtain when the multiplier adjustment is used to find $\mu$ is dependent on the quality of the schedule $S$. Initially, $S$ is obtained by applying the heuristic method that is presented in the next section. However, whenever a better solution is found in the branch and bound algorithm, $S$ is updated and multipliers are computed from this new schedule. Thus, the multipliers change only when an improved upper bound is found.

The second approach for determining values of the multipliers uses *subgradient optimization.* Let $\mu^{(l)}$ be a vector of multipliers at iteration $l$ of this procedure, and let $q_t^{(l)}$ denote the number of families for which a set-up or some processing of a job is scheduled in the interval $[t-1, t]$ in the computation of $\mathrm{LB}(\mu^{(l)})$. Initially, we set $\mu^{(1)}$ to be the multipliers obtained by the multiplier adjustment method, where $S$ is obtained by first forming a single batch from each family and then sequencing these batches in non-decreasing order of their WPT values. Thereafter, we set

$$\mu_t^{(l+1)} = \max\left\{ \mu_t^{(l)} + \lambda^{(l)} \frac{(\mathrm{UB} - \mathrm{LB}(\mu^{(l)}))(q_t^{(l)} - 1)}{\sum_{t=1}^{T} (q_t^{(l)} - 1)^2}, 0 \right\},$$

where UB is an upper bound on the minimum value of the total weighted completion time and $\lambda^{(l)}$ is a scalar step length which satisfies $0 \leq \lambda^{(l)} \leq 2$. At the root node of the branch and bound search tree, we set $\lambda^{(1)} = 2$, and perform 20 subgradient optimization iterations, halving the step length when 5 successive iterations fail to improve the lower bound. For other nodes of the search tree, we set $\mu^{(1)}$ to be the vector of multipliers at the parent node, we set $\lambda^{(1)} = 0.1$, and perform 3 subgradient optimization iterations, halving the step length when no improvement in the lower bound is observed for 2 successive iterations.

## 4 A heuristic method

In this section, we propose a heuristic method which is applied at the root node of the branch and bound search tree to find an upper bound UB on the minimum value of the total weighted completion time. Our heuristic uses a constructive procedure, which is based on observations of Gupta [6], to find an initial schedule. Descent procedures are then applied with the aim of reducing the total weighted completion time. A descent method attempts to improve on a current solution by searching, in some suitably defined neighbourhood, for a new solution which has a lower objective function value. If such an improved solution is found, it becomes the current solution from which further improvement is sought. If no improvement is possible, then the method terminates and the solution is a local optimum.

**Heuristic method**

*Step 1.* Index the jobs within each family in SWPT order. Set $a_f = (s_f + p_{1f})/w_{1f}$ and $m_f = 1$ for $f = 1, \ldots, F$. Set $k = 1$ and $g = 0$ (where $k$ is the first unfilled position in the sequence and $g$ is the family to which the last scheduled job belongs).

*Step 2.* Choose family $f$ such that $a_f$ is as small as possible. Schedule job $(m_f, f)$ in position $k$, set $m_f = m_f + 1$ and $k = k + 1$. If $m_f > n_f$, set $a_f = \infty$; otherwise, set $a_f = p_{m_f,f}/w_{m_f,f}$. If $g = 0$, set $g = f$. If $g \neq f$, set $a_g = (s_g + p_{m_g,g})/w_{m_g,g}$ and $g = f$.

*Step 3.* If $k \leq N$, then go to Step 2.

*Step 4.* Apply *shift-batch*: a descent algorithm which uses a neighbourhood based on the interchange of adjacent pairs of batches.

*Step 5.* Apply *shift-job*: a descent algorithm where the elements of the neighbourhood are constructed as follows. For each batch, its last job is shifted backwards to the start of the next batch containing jobs of the same family (unless it is the last batch of a family), or its first job is shifted forwards to the end of previous batch containing jobs of the same family (unless it is the first batch of a family). The last job of the last batch of a family is also shifted backwards, thus creating a new batch.

*Step 6.* Apply shift-batch.

The schedule generated by the heuristic method is consistent with Theorem 1 as it has the property that the jobs within each family are sequenced in SWPT order.

## 5 Branch and bound algorithms

In this section, we describe three branch and bound algorithms. The first is based on forward branching with objective splitting (FBOS), and is essentially the algorithm of Mason and Anderson [10]. At the root node of the branch and bound search tree, we apply the preprocessing routine that is described in Section 2 to form composite jobs and thereby reduce the problem size. Also, the heuristic method of Section 4 is applied to generate an initial upper bound. At each node of the branch and bound search tree, Rules 1 to 5 of Section 2 are used as pruning devices (Rule 3 is not used in the original algorithm of Mason and Anderson), together with the dynamic programming dominance rule. A depth-first search strategy is used, where the immediate descendants of a node are explored in non-decreasing order of their lower bounds.

The second branch and bound algorithm employs forward branching and multiplier adjustment (FBMA). Apart from the method of computing lower bounds, it is identical to the first algorithm. Initially, the lower bound LB($\mu$) uses multipliers that are obtained from the heuristic schedule. Within the computation of the lower bound, an upper bound is also computed: form batches from the jobs of the same family that are scheduled contiguously in the lower bounding computation, use Theorem 2 to sequence these batches, and evaluate the resulting schedule. If the resulting upper bound is less than the current upper bound, then UB is updated, and multipliers are based on this improved schedule in subsequent lower bounding computations.

Our third branch and bound algorithm uses a binary branching rule and subgradient optimization (BBSO). At the root node, the preprocessing routine of Section 2 is applied. At each node of the search tree, we compute an upper bound by constructing two heuristic schedules. The first heuristic forms a batch from each family and then sequences these batches in non-decreasing order of their WPT values. In the second heuristic, jobs are sequenced in SWPT order. The Lagrangean lower bound is computed at each node, as described in Section 3, where subgradient optimization is used to find values of the multipliers. Our branching rule selects a pair of jobs $(j-1, f)$ and $(j, f)$, where $(j, f) \in \mathcal{N}$ and $j > 1$, and constrains these jobs either to be contained in the same batch, or to be contained in different batches, Unlike forward branching, this binary branching allows the possibility of making important decisions at the higher levels of the branch and bound search tree. For the branch in which jobs $(j-1, f)$ and $(j, f)$ must be contained in the same batch, we form a composite job from $(j-1, f)$ and $(j, f)$. In the other branch in which job $(j, f)$ starts a new batch, we partition the jobs of family $f$ into two families containing jobs $(1, f), \ldots, (j-1, f)$ and jobs $(j, f), \ldots, (n_f, f)$, respectively. We select job $(j, f)$ so that the Lagrangean problem schedules jobs $(j-1, j)$ and $(j, f)$ to complete at time $t'$ and $t$, respectively, where $t' + p_{jf} < t$ and $z(j, t' + p_{jf}) - z(j, t)$ is as large as possible ($z(j, t' + p_{jf}) - z(j, t)$ is obtained from the dynamic programming recursion for the Lagrangean problem and provides a measure of the cost of not splitting family $f$ so that job $(j, f)$ starts a new batch). For this binary branching rule, initial experiments with our Lagrangean lower bounds, where multipliers are obtained using subgradient optimization and using the multiplier adjustment method, indicate that the subgradient optimization approach is superior. As in the two other algorithms, a depth-first search strategy is used.

## 6 Computational experience

In this section, we report on the results of computational tests to assess the effectiveness of the three branch and bound algorithms. Algorithms FBOS and FBMA were coded in ANSI-C, whereas BBSO was coded in FORTRAN 77. All algorithms were run on a HP 9000/715 computer. For FBOS, computation is abandoned if a limit of 50 000 nodes is exceeded, whereas a time limit of 600 seconds is used for BBSO. No limits are applied for FBMA (all problems can be solved using reasonable computational resources).

Test problems with 30, 40 and 50 jobs, and with 4, 6, 8 and 10 families were generated as follows. For each combination of $N$ and $F$, the jobs are uniformly distributed across families, so that each family contains either $\lfloor N/F \rfloor$ or $\lceil N/F \rceil$ jobs. Processing times and weights are randomly generated integers from the uniform distribution defined on $[1,10]$. Since the size of set-up times relative to processing times may affect problem 'hardness', we generated problems with small (S), medium (M) and large (L) set-up times. Medium set-up times are randomly generated integers from the uniform distribution defined on $[1,10]$. Having generated an instance with medium set-up times $s_f$ $(f = 1,\ldots,F)$, corresponding instances with small set-up times $\lceil s_f/2 \rceil$ and with large set-up times $2s_f$ were constructed. For each of the 12 combinations of $N$ and $F$, 50 test problems with small, medium and with large set-up times were created.

Our test problems cover a range of different scenarios. The number of jobs per family ranges from 3 (for $N = 30$ and $F = 10$) to 12 and 13 (when $N = 50$ and $F = 4$). Also, a variety of set-up time ranges is considered. Problems with very small set-up times are likely to exert little influence, so that an optimal sequence of jobs will be close to SWPT. On the other hand, batches will be formed from complete families when set-up times are very large. Both of these extreme cases are likely to be relatively easy to solve. Possible schedules in our instances may involve the machine undergoing a setup time for only about 4% of the time (for $N = 50$ and $F = 4$ when set-up times are small and there is only one set-up for each family), or for about 67% of the time (for large set-up times when a set-up time is incurred for every job).

Table 1 gives computational results for the three branch and bound algorithms. For each combination of $N$ and $F$, average computation times in seconds (ACT), average numbers of nodes (ANN) and numbers of unsolved problems (NU) are listed for the instances with small, medium and large set-up times. When there are unsolved problems, the values listed under ACT and ANN are lower bounds on the true averages.

We first observe from Table 1 that there are significant numbers of unsolved problems for algorithms FBOS and BBSO. However, FBMA solves all test problems without generating large search trees or requiring excessive computation time. For instance, among the 600 test problems with medium set-up times, only 6 require more than 3 minutes of computation time.

The computational effort varies significantly with $F$ for algorithms FBOS and BBSO, but is fairly stable for FBMA. Algorithm FBOS captures some of the problem structure through its dominance rules, thus enabling it to solve the problems with a small number of families. However, the objective splitting bound is too weak to control the explosion in tree size for larger problems, especially when there are many families. Algorithm BBSO also captures part of the problem structure by its use of a binary branching rule that splits the 'right' family. This algorithm is clearly

Table 1. Comparative computational results.

| Set-up | | | FBOS | | | FBMA | | BBSO | | |
|---|---|---|---|---|---|---|---|---|---|---|
| times | $N$ | $F$ | ACT | ANN | NU | ACT | ANN | ACT | ANN | NU |
| S | 30 | 4 | 0.1 | 657 | – | 1.9 | 74 | 24.6 | 440 | – |
| | | 6 | 0.4 | 1775 | – | 2.7 | 92 | 9.1 | 113 | – |
| | | 8 | 2.0 | 3732 | – | 2.7 | 95 | 9.0 | 103 | – |
| | | 10 | 1.0 | 2778 | – | 1.6 | 66 | 5.5 | 53 | – |
| | 40 | 4 | 0.7 | 2477 | – | 11.2 | 184 | 211.1 | 2401 | 7 |
| | | 6 | 7.5 | 9096 | – | 11.8 | 161 | 119.8 | 1127 | 5 |
| | | 8 | 41.0 | 22167 | 7 | 12.0 | 171 | 60.4 | 507 | 1 |
| | | 10 | 37.0 | 20292 | 7 | 9.5 | 148 | 34.6 | 277 | 1 |
| | 50 | 4 | 3.4 | 5591 | – | 34.5 | 365 | 448.4 | 3554 | 34 |
| | | 6 | 68.5 | 29478 | 9 | 56.0 | 437 | 305.9 | 2124 | 17 |
| | | 8 | 112.0 | 44658 | 35 | 69.0 | 451 | 227.8 | 1334 | 9 |
| | | 10 | 112.0 | 46818 | 42 | 61.0 | 436 | 160.4 | 891 | 7 |
| M | 30 | 4 | 0.1 | 371 | – | 1.6 | 44 | 7.9 | 108 | – |
| | | 6 | 0.2 | 757 | – | 1.9 | 53 | 5.5 | 64 | – |
| | | 8 | 0.3 | 1199 | – | 1.4 | 43 | 3.3 | 28 | – |
| | | 10 | 0.3 | 956 | – | 1.3 | 45 | 3.2 | 26 | – |
| | 40 | 4 | 0.4 | 1522 | – | 11.3 | 165 | 119.7 | 1243 | 5 |
| | | 6 | 1.9 | 4082 | – | 10.0 | 112 | 47.6 | 401 | 1 |
| | | 8 | 10.5 | 8703 | 1 | 11.6 | 143 | 37.8 | 298 | – |
| | | 10 | 6.0 | 6992 | – | 7.1 | 94 | 13.2 | 85 | – |
| | 50 | 4 | 1.4 | 3456 | – | 31.0 | 217 | 331.0 | 2528 | 2 |
| | | 6 | 23.5 | 15226 | – | 46.5 | 263 | 149.9 | 937 | 3 |
| | | 8 | 67.0 | 29040 | 18 | 54.0 | 281 | 109.6 | 593 | 3 |
| | | 10 | 79.0 | 33810 | 24 | 32.0 | 200 | 92.2 | 485 | 1 |
| L | 30 | 4 | 0.1 | 198 | – | 1.4 | 30 | 5.5 | 69 | – |
| | | 6 | 0.1 | 294 | – | 1.0 | 22 | 2.2 | 19 | – |
| | | 8 | 0.1 | 322 | – | 0.8 | 22 | 2.1 | 16 | – |
| | | 10 | 0.1 | 257 | – | 0.8 | 21 | 2.0 | 14 | – |
| | 40 | 4 | 0.1 | 661 | – | 7.8 | 65 | 38.4 | 354 | 1 |
| | | 6 | 0.3 | 1292 | – | 5.8 | 51 | 13.2 | 95 | – |
| | | 8 | 0.4 | 1655 | – | 4.8 | 49 | 12.1 | 83 | – |
| | | 10 | 0.6 | 1770 | – | 2.8 | 34 | 5.5 | 29 | – |
| | 50 | 4 | 0.4 | 1535 | – | 24.7 | 113 | 126.6 | 845 | 5 |
| | | 6 | 4.1 | 5215 | – | 31.5 | 180 | 103.7 | 620 | 3 |
| | | 8 | 10.7 | 8225 | 2 | 23.5 | 118 | 18.2 | 88 | – |
| | | 10 | 18.3 | 11633 | 2 | 18.1 | 101 | 23.4 | 105 | – |

FBOS: algorithm with forward branching and objective splitting.
FBMA: algorithm with forward branching and multiplier adjustment.
BBSO: algorithm with binary branching and subgradient optimization.
ACT: average computation time in seconds on a HP 9000/715 computer.
ANN: average number of nodes in the branch and bound search tree.
NU: number of unsolved problems (out of a total of 50).

quite effective when there are many small families. With few large families, however, there are many ways to split each family, and the Lagrangean bound in combination with subgradient optimization is not strong enough to restrict the size of the search tree. The disadvantage of BBSO is the fact that it fails to take advantage of the problem structure as embedded in the dominance rules. Algorithm FBMA exhibits a stable behaviour. It combines the efficient exploitation of dominance rules within the search tree with a quick implementation of a reasonably good Lagrangean bound. The result is that all problems can be solved quite effectively, and the algorithm is not very sensitive to the relative size of $N$ and $F$. Dominance Rule 3 is particularly effective for problems with larger numbers of families. As an illustration, for the original implementation of algorithm FBOS by Mason and Anderson [10], which does not use Rule 3, problems with $F = 8$ and $F = 10$ are much more demanding on computer resources compared with our version which includes Rule 3. Without Rule 3 in algorithm FBOS, problems with $F = 10$ are harder to solve than those with $F = 8$. The use of Rule 3 helps to overcome the inability of the objective splitting lower bound to restrict the search for large $F$.

As expected, the relative size of set-up times affects problem hardness. Results in Table 1 indicate that all algorithms find the test problems with small set-up times to be the hardest, and those with large set-up times are the easiest. With large set-up times, family splitting is very expensive. As a result, the splitting of families into batches in an optimal solution is limited, and the combinatorial characteristics of the problem are reduced. As set-up times become smaller, more family splits become potentially attractive, and the problem achieves its full combinatorial complexity. Algorithms FBOS and BBSO experience difficulty in coping with this complexity in our test problems, although FBMA performs adequately.

In terms of average computation times, algorithm FBOS dominates BBSO and FBMA for problems with a few large families. Its 'quick and dirty' lower bound is more efficient than the more time consuming Lagrangean method for these easier problems. For the harder problems, algorithm FBMA is clearly superior to both FBOS and BBSO.

Having established that algorithm FBMA is superior to the other branch and bound algorithms, we report on the results of some further tests with FBMA. We use test problems with 60 and 70 jobs and with 4, 8 and 15 families that are generated in a similar way to the other problems. These additional tests were run on a HP9000/G50 computer, which is about twice as fast as the HP 9000/715 on which the previous results are obtained. A time limit of 300 seconds for algorithm FBMA is used. Table 2 shows that algorithm FBMA becomes rather time consuming for 60- and 70-job problems. This is mainly due to the dynamic programming recursion within the lower bound computation. The table also gives an indication of the number of batches in the optimal solution, i.e., of the number of times families are split on average.

## 7 Concluding remarks

This paper considers the problem of scheduling families of jobs on a single machine to minimize the total weighted completion time, where a set-up time is incurred whenever the machine switches from processing a job in one family to a job in another family. A new lower bounding scheme

Table 2. Computational results for FBMA on larger problems.

| Set-up | | | FBMA | | | No. of batches | |
|---|---|---|---|---|---|---|---|
| times | $N$ | $F$ | ACT | ANN | NU | Min | Max |
| S | 60 | 4 | 41.7 | 429 | – | 10 | 15 |
| | | 8 | 97.6 | 715 | 3 | 16 | 25 |
| | | 15 | 67.2 | 645 | 2 | 24 | 32 |
| | 70 | 4 | 93.4 | 611 | 1 | 11 | 17 |
| | | 8 | 175.7 | 976 | 15 | 18 | 26 |
| | | 15 | 144.4 | 811 | 8 | 27 | 36 |
| M | 60 | 4 | 35.0 | 268 | – | 8 | 14 |
| | | 8 | 61.6 | 405 | 1 | 14 | 22 |
| | | 15 | 28.0 | 296 | – | 22 | 29 |
| | 70 | 4 | 82.5 | 399 | 1 | 10 | 14 |
| | | 8 | 149.4 | 637 | 8 | 15 | 23 |
| | | 15 | 107.1 | 535 | 8 | 24 | 30 |
| L | 60 | 4 | 32.9 | 163 | – | 7 | 11 |
| | | 8 | 33.6 | 173 | – | 13 | 18 |
| | | 15 | 10.3 | 82 | – | 17 | 25 |
| | 70 | 4 | 70.8 | 224 | – | 8 | 11 |
| | | 8 | 108.1 | 361 | 6 | 13 | 20 |
| | | 15 | 37.9 | 148 | – | 20 | 27 |

FBMA: algorithm with forward branching and multiplier adjustment.
ACT: average computation time in seconds on a HP 9000/G50 computer.
ANN: average number of nodes in the branch and bound search tree.
NU: number of unsolved problems (out of a total of 50).

based on a Lagrangean relaxation of machine capacity constraints is derived. Also, a multiplier adjustment method for obtaining quickly computed values of the multipliers is proposed.

Computational results show that when the Lagrangean multiplier adjustment bound is used in a branch and bound algorithm which adopts a forward branching rule and incorporates various dominance rules (algorithm FBMA), problems with 70 jobs can be solved using reasonable computational resources. This algorithm is far superior to the branch and bound algorithm of Mason and Anderson [10] (algorithm FBOS) which uses objective splitting in its lower bounding scheme. Computational tests are also reported for another algorithm which uses a Lagrangean subgradient optimization bound and which employs a more flexible branching rule (algorithm BBSO). The dominance rules cannot be used with this branching rule. Results indicate that the increased flexibility offered by the branching rule does not compensate for extra computation that is incurred due to the inapplicability of the dominance rules.

Our computational results provide further evidence to support the view that, in a Lagrangean bounding scheme, a multiplier adjustment method for determining values of the multipliers is often preferable to the computationally expensive subgradient optimization method.

Scheduling problems which involve batching have both theoretical interest and practical importance. Studies which develop and test algorithms for solving such problems are relatively scarce. However, for minimizing total weighted completion time on a single machine, this paper, together with the study of Crauwels et al. [3] which develops various local search heuristics, provides a thorough treatment of the problem. An important area for future research is to design enumerative algorithms and local search heuristics for single machine problems with other objectives, and to consider multi-machine models.

## Acknowledgements

This research was partly supported by NATO Collaborative Research Grant 0224/88 and by the International Association for the Promotion of Cooperation with Scientists from the Independent States of the Former Soviet Union, INTAS-93-257.

## References

[1] B.-H. Ahn and J.-H. Hyun, "Single facility multi-class job scheduling", *Computers & Operations Research* 17 (1990) 265–272.

[2] H. Belouadah and C.N. Potts "Scheduling identical parallel machines to minimize total weighted completion time", *Discrete Applied Mathematics* 48 (1994) 201–218.

[3] H.A.J. Crauwels, C.N. Potts and L.N. Van Wassenhove "Local Search Heuristics for Single Machine Scheduling with Batch Set-up Times", Working Paper 94/07/TM, INSEAD, Fontainebleau, France, 1994.

[4] M.L. Fisher, "A dual algorithm for the one-machine scheduling problem", *Mathematical Programming* 11 (1976) 229–251.

[5] J.B. Ghosh, "Batch scheduling to minimize total completion time", *Operations Research Letters* 16 (1994) 271–275.

[6] J.N.D. Gupta, "Single facility scheduling with multiple job classes", *European Journal of Operational Research* 8 (1988) 42–45.

[7] A.M.A. Hariri and C.N. Potts "An algorithm for single machine sequencing with release dates to minimize total weighted completion time", *Discrete Applied Mathematics* 5 (1983) 99–109.

[8] E.L. Lawler, "Sequencing jobs to minimize total weighted completion time subject to precedence constraints", *Discrete Applied Mathematics* 2 (1978) 75–90.

[9] A.J. Mason " Genetic Algorithms and Scheduling Problems", Ph.D. dissertation, Department of Engineering, University of Cambridge, U.K., 1992.

[10] A.J. Mason and E.J. Anderson, "Minimizing flow time on a single machine with job classes and setup times", *Naval Research Logistics* 38 (1991) 333–350.

[11] C.L. Monma and C.N. Potts, "On the complexity of scheduling with batch setup times", *Operations Research* 37 (1989) 798–804.

[12] C.N. Potts and L.N. Van Wassenhove, "An algorithm for single machine sequencing with deadlines to minimize total weighted completion time", *European Journal of Operational Research* 12 (1983) 379–387.

[13] C.N. Potts and L.N. Van Wassenhove, "A branch and bound algorithm for the total weighted tardiness problem", *Operations Research* 33 (1985) 363–377.

[14] C.N. Potts and L.N. Van Wassenhove, "Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity", *Journal of the Operational Research Society* 43 (1992) 395–406.