JOURNAL OF SCHEDULING
J. Sched. 2000; **3**:51–69

Lower bounds and algorithms for flowtime minimization on a single machine with set-up times

Simon Dunstall[1,†], Andrew Wirth[1,*,‡] and Kenneth Baker[2,§]

[1] *Department of Mechanical and Manufacturing Engineering, University of Melbourne, Parkville 3052, Australia*
[2] *Tuck School, Dartmouth College, Hanover, NH 03755, U.S.A.*

SUMMARY

We consider the scheduling of N jobs divided into G families for processing on a single machine. No set-up is necessary between jobs belonging to the same family. A set-up must be scheduled when switching from the processing of family i jobs to those of another family j, $i \neq j$, the duration of this set-up being the sequence-independent set-up time s_j for family j. We propose lower bounds for the problem of minimizing the weighted flowtime on a single machine with family set-up times and static job availability. These lower bounds are incorporated into a branch-and-bound algorithm which can efficiently solve instances with up to 70 jobs. Copyright © 2000 John Wiley & Sons, Ltd.

KEY WORDS: single-machine scheduling; family set-up times; branch and bound

1. INTRODUCTION

A recent trend in the analysis of scheduling models integrates batching decisions with sequencing decisions. The interplay between batching and sequencing reflects the realities of the small-volume, high-variety manufacturing environment and adds a new feature to traditional scheduling problems. Practical interest in this topic has given rise to new research efforts, and there has been a series of articles in the research literature surveying the rapidly developing state of knowledge. Examples include Ghosh [1], Liaee and Emmons [2], Potts and Van Wassenhove [3], and Webster and Baker [4].

This paper deals with an important theoretical and practical problem in this area. We examine the single-machine model with *family* (or *group*) set-up times and a criterion of minimizing total weighted job completion time (weighted flowtime). We propose new lower bounds for this problem, and then turn our attention to refinement of a previously proposed branch-and-bound algorithm. The benefits of our refinements are illustrated by computational experiments.

*Correspodence to: Andrew Wirth, Department of Mechanical and Manufacturing Engineering, Univeristy of Melbourne, Parkville, Victoria 3052, Australia
† E-mail: simondun@mame.mu.oz.au
‡ E-mail: wirth@mame.mu.oz.au
§ E-mail: ken.baker@dartmouth.edu

2. PROBLEM STATEMENT AND LITERATURE REVIEW

It is commonly proposed in the literature that the set of jobs to be scheduled can be successfully partitioned, according to processing requirements, into mutually exclusive and exhaustive families. It is assumed that the jobs have the property that a set-up need only be scheduled when switching from a job of one family to a job belonging to a different family. This property enables us to take advantage of the 'group nature' of the jobs and avoid set-up time penalties by sequencing jobs of the same family successively, in *batches.* We assume in this paper that the set-up times are sequence independent, that is, the set-up time between batches depends only upon the family being switched to. We also assume that an initial set-up for the machine is required, the duration of this initial set-up being equal to the set-up time for the relevant family.

We denote by G the number of families in an instance and N the total number of jobs in an instance. The number of jobs belonging to family i $(1\leqslant i\leqslant G)$ is N_i, with $\sum_{i=1}^{G} N_i = N$. The set-up time for family i is represented by the symbol s_i, this being a non-negative integer. The jth job of family i, $a_{i[j]}$ $(1\leqslant j\leqslant N_i)$, is assigned a non-negative integer processing time $p_{i[j]}$ and positive integer weight $w_{i[j]}$. All jobs are considered to be indivisible and to be available from time zero. With a schedule determined, the completion time of $a_{i[j]}$ is denoted by $C_{i[j]}$.

Jobs of the same family may appear in one or more batches in a 'good' schedule for an instance of our problem. The number of batches in a schedule is denoted by b, with the kth batch represented by the symbol B_k. The first job of a batch is immediately preceded by a set-up. The set-up time for a batch B_k is given by σ_k, the number of jobs in B_k is given by λ_k, and the jth job of B_k, job $\alpha_{k[j]}$, has processing time and weight denoted by $\tau_{k[j]}$ and $\varpi_{k[j]}$, respectively. The total time required to process a batch is T_k, where $T_k = \sigma_k + \sum_{j=1}^{\lambda_k} \tau_{k[j]}$, while the total weight of a batch is given by $W_k = \sum_{j=1}^{\lambda_k} \varpi_{k[j]}$. The *weighted mean processing time* WMPT(B_k) of a batch B_k is defined as T_k/W_k.

In this paper we adopt the classification scheme described by Lawler *et al.* [5]. In this scheme a problem is represented by $\alpha|\beta|\gamma$, where α describes the machine environment, β identifies job properties which vary from those commonly assumed and γ indicates the objective function. Within this paper we extend the β field of this scheme to cater for problems with set-up times, using the symbol s_{ij} to indicate sequence-dependent set-up times and s_i to indicate sequence-independent set-up times. For example, our problem is denoted by $1|s_i|\sum wC$ in this classification scheme.

The $1|s_i|\sum wC$ problem and related problems have been studied by a number of researchers. Analysis of problem structure has been provided by Mason and Anderson [6] and Monma and Potts [7]. The optimality conditions developed by these authors are reviewed in Section 3. The $1|s_{ij}|\sum wC$ problem is optimally solvable in $O(G^2N^G)$ time using the dynamic programming approach of Ghosh [1], this time being polynomial in N for a fixed number of families. The computational complexity of the $1|s_i|\sum C$ problem with an arbitrary number of families remains an important open question, yet the $1|s_{ij}|\sum C$ problem is known to be NP-hard [8]. The dynamic $1|r_j, s_i|\sum C$ problem is clearly NP-hard, as the corresponding problem without set-up times is (again, see Reference [8]). This result is true even when each job has the same ready time as every other job of its family.

Monma and Potts developed a dynamic program for $1|s_{ij}|\sum wC$ running in $O(G^2N^{G^2+G})$ time, and Potts [9] adapted this algorithm to provide an $O(N^3)$ algorithm for the problem with two families only. Ghosh [1] proposed a more efficient $O(G^2N^G)$ dynamic program for the same problem, closely resembling the earlier dynamic programs developed by Psaraftis [10] (identical jobs within each family), Ahn and Hyun [11] ($1|s_{ij}|\sum C$) and Bruno and Sethi [12] ($1|s_i|\sum wC$).

The branch-and-bound algorithms developed by Mason and Anderson and by Crauwels *et al.* [13] for the $1|s_i|\sum wC$ problem utilized a strong set of dominance conditions derived from the known optimality conditions for the problem. The most successful algorithm of Crauwels *et al.* is similar in structure to the Mason and Anderson algorithm, but utilizes additional dominance rules and a strong lower bound based upon a Lagrangian relaxation of the problem. Computational testing of the Crauwels *et al.* algorithm [13, 14] established that, in comparison to Mason and Anderson's original procedure, it has the ability to successfully solve instances with much greater numbers of jobs.

The Lagrangian lower bound is hampered by a time-complexity function which depends not only upon the number of jobs in an instance but also upon the duration, in time units, of a schedule for the instance. As a result, its use can only be recommended when the maximum set-up and processing times in the instance are restricted to values such as 10 as utilized by Crauwels *et al.* It is therefore clear that development and substitution of a strong lower bound whose running time is independent of the duration of a schedule will allow problems of increased practical significance, i.e. those with greater maximum set-up and processing times, to be optimally solved in reasonable time. This is the main goal of our paper.

Heuristics applicable to single-machine flowtime problems with family set-up times have been developed by a number of workers. Ahn and Hyun [11] and Gupta [15] have proposed heuristics for the $1|s_{ij}|\sum C$ problem, and Crauwels *et al.* [16] devised a range of local search heuristics for the $1|s_i|\sum wC$ problem. Crauwels *et al.* presented an extensive computational study which compared the performance of their heuristics, the Ahn and Hyun heuristic (modified to address the $1|s_i|\sum wC$ problem), and a genetic algorithm proposed by Mason [17] for the same problem. Their testing suggested that a tabu search heuristic was most effective on average, although Mason's algorithm was shown to outperform the tabu search procedure for instances with larger numbers of families.

Crauwels [14] subsequently investigated a series of heuristics with a binary-based representation of sequences. This representation is similar but not identical to that utilized within Mason's genetic algorithm. The most successful of these binary-based heuristics was a multi-start descent heuristic. Computational experience reported by Crauwels shows that, for many instances with up to 100 jobs, the binary-based descent heuristic delivered superior performance compared to alternative algorithms, including the tabu search heuristic of Reference [16].

Local search heuristics for the $1|s_i|\sum wC$ problem have also been devised by Baker [18] and Dunstall [19], while Williams and Wirth [20] have developed a heuristic for the $1|s_i|\sum C$ problem. It can be noted that the heuristic reported by Williams and Wirth cannot, in fact, be guaranteed to provide schedules satisfying all three major optimality conditions for the $1|s_i|\sum C$ problem, contrary to a claim made in their paper (see Reference [19]). Nowicki and Zdrzalka [21] have proposed a tabu-search heuristic which can be applied to minimize any regular objective function on a single machine with major sequence-independent set-up times and minor set-up times (between sub-families) which take a constant value for all sub-families. Nowicki and Zdrzalka evaluated their heuristic computationally for the objectives of minimizing maximum weighted lateness and total weighted tardiness. Finally, Liao and Liao [22] have addressed the mean flowtime problem with major and minor sequence-independent set-up times. Their heuristic approach, based on a dynamic programming solution of sub-problems, can be computationally expensive yet has the ability to solve well many instances of this practically significant problem.

The process of forming families is typically not addressed by scheduling research. However, we note that in applications of group technology (GT), the tooling analysis sub-technique of production flow analysis can be used to form families (i.e. tooling families) by grouping jobs with similar processing features [23]. This observation highlights a link between the scheduling problems discussed in this paper and the scheduling of facilities where the GT concepts have been applied.

The sequence-independent set-up times model is restrictive in comparison to the more general sequence-dependent set-up times model, but has the benefit, from a scheduling point of view, of yielding problems which are significantly easier to solve. Within the scheduling literature practical examples of production systems involving sequence-independent set-up times are not common (but see References [3, 6]). This state of affairs may reflect a general trend for this model to be unduly restrictive for in-practice application.

The additive changeovers model, proposed by Sule [24], may in a number of cases represent an effective compromise between the simple sequence-independent set-up times model and the complex sequence-dependent set-up times model. In this enhanced model both set-up times and teardown (set-down) times are allowed for each family, each of these times being sequence-independent. Fortuitously, many single-machine problems with additive changeovers can be solved as equivalent problems with sequence-independent set-up times only [25]. The problem addressed in this paper is one such example, as shown by Mason and Anderson.

3. OPTIMALITY CONDITIONS AND DOMINANCE RELATIONSHIPS

Three necessary, but not sufficient, optimality conditions have been identified for the $1|s_i|\sum wC$ problem. The first of these optimality conditions was developed by Monma and Potts [7].

Theorem 1. There exists an optimal sequence for the $1|s_{ij}|\sum wC$ *problem* (*where set-ups satisfy the triangle inequality*) *within which jobs of the same family appear in SWPT order*, *i.e. if*

$$\frac{p_{i[j]}}{w_{i[j]}} \leqslant \frac{p_{i[k]}}{w_{i[k]}}$$

then there exists an optimal sequence within which $a_{i[j]}$ *precedes* $a_{i[k]}$.

Theorem 1 can be conveniently referred to as the '*SWPT-within-families rule*' for jobs. Throughout this paper we will assume, without loss of generality, that jobs are indexed within their families in SWPT-order (i.e. indexed in accordance with the SWPT-within-families rule).

The next optimality condition (Theorem 2), attributable to Mason and Anderson [6], we term the '*SWMPT rule*' for batches.

Theorem 2. There exists an optimal sequence for the $1|s_i|\sum wC$ *problem within which batches appear in shortest weighted mean processing time order*, *i.e.*

$$\mathrm{WMPT}(B_k) = \frac{T_k}{W_k} \leqslant \frac{T_{k+1}}{W_{k+1}} = \mathrm{WMPT}(B_{k+1})$$

for $1 \leqslant k < b$.

The work sequenced between any two batches B_u and B_v of the same family can be viewed as a *composite job* with processing time $\sigma_v + \sum_{k=u+1}^{v-1} T_k$ and weight $\sum_{k=u+1}^{v-1} W_k$. Theorem 3 (Mason and Anderson) represents an '*extended SWPT rule*': the jobs belonging to a family and the composite jobs between the batches of this family follow SWPT order in an optimal sequence.

Theorem 3. There exists an optimal sequence for the $1|s_i|\sum wC$ *problem within which jobs belonging to different batches of the same family satisfy*

$$\frac{\tau_{u[\lambda_u]}}{\varpi_{u[\lambda_u]}} \leqslant \frac{\sigma_v + \sum_{k=u+1}^{v-1} T_k}{\sum_{k=u+1}^{v-1} W_k} \leqslant \frac{\tau_{v[1]}}{\varpi_{v[1]}}$$

where $1 \leqslant u < v \leqslant b$.

Using the above optimality conditions, Mason and Anderson formed a set of strong dominance rules applicable to the solution of the $1|s_i|\sum wC$ problem using branch-and-bound methods. Crauwels *et al.* added a further rule to this set (Rule 5 below). The five dominance rules utilized by Crauwels *et al.* are given as Rules 1–5. A new rule, Rule 6, is also included in this list.

(1) In an optimal sequence for the $1|s_i|\sum wC$ problem, where B_u and B_v are batches of the same family $(u < v)$

$$\text{WMPT}(B_u) \leqslant \frac{\tau_{v[1]}}{\varpi_{v[1]}}$$

Thus if extending a partial sequence with the addition of a new batch of some family i, we should add jobs to this batch at least until we use up all family i jobs (i.e. $a_{i[N_i]}$ is scheduled) or the next unscheduled job of this family has a weighted processing time greater than the current weighted mean processing time of the batch.

(2) Consider a partial sequence S within which the last batch is B_v, the last scheduled job is $a_{i[j]}$, and $\text{WMPT}(B_{v-1}) > \text{WMPT}(B_v)$. If $j < N_i$, job $a_{i[j+1]}$ should be added to the end of B_v. If $j = N_i$ the partial sequence S cannot satisfy Theorem 2 and is suboptimal.

(3) In an optimal sequence for the $1|s_i|\sum wC$ problem, where B_u and B_v are batches of the same family $(u < v)$

$$\frac{\tau_{u+1[1]}}{\varpi_{u+1[1]}} \leqslant \frac{\tau_{v[1]}}{\varpi_{v[1]}}$$

Thus if we are extending a partial sequence S by adding a new batch B_{u+1}, and S currently terminates with a batch B_u of some family i, the weighted processing time $\tau_{u+1[1]}/\varpi_{u+1[1]}$ of the first job $\alpha_{u+1[1]}$ of B_{u+1} can be no greater than the weighted processing time of the next unscheduled job of family i.

(4) If we wish to extend a partial sequence S by starting a new batch B_v of family i, we must ensure that the two inequalities of Theorem 3 will be satisfied between B_v and the latest scheduled batch B_u of the same family in S (if B_u exists).

(5) Consider a partial sequence S, and let the last batch in S be B_v, this batch containing jobs of some family i. Let n_k be the number of jobs of family k which appear in S, $0 \leqslant n_k \leqslant N_k$, and assume without loss of generality that these are the first n_k jobs of family k. Let a family $k \neq i$ belong to set U if $n_k < N_k$, this being the set of families which currently contain

unscheduled jobs. If partial sequence S is such that

$$\frac{T_v}{W_v} > \min_{k \in U} \left\{ \frac{s_k + \sum_{j=n_k+1}^{N_k} p_{k[j]}}{\sum_{j=n_k+1}^{N_k} w_{k[j]}} \right\}$$

then if $n_i < N_i$, job $a_{i[n_i+1]}$ must be appended next to S. If $n_i = N_i$, partial sequence S cannot be optimal.

(6) Consider a partial sequence S within which the last batch of some family i is batch B_u. Let family i be such that at least one family i job remains unscheduled; let this be $a_{i[j]}$. Let the last batch in the partial sequence be batch B_b, this batch constructed according to Rules 1 and 2. From Theorems 1–3, if the partial sequence is to be part of an optimal sequence, the following inequality must hold:

$$\frac{s_i + \sum_{k=u+1}^{b} T_k}{\sum_{k=u+1}^{b} W_k} \leqslant \frac{p_{i[j]}}{w_{i[j]}} \tag{1}$$

otherwise it is not possible for $a_{i[j]}$ to be sequenced at any position after B_b and satisfy Theorem 3. This is because the value of the left-hand side of (1) cannot decrease through addition (to the end of S) of unscheduled jobs from any family.

Rule 6 has its basis in the 'extended SWPT rule'. It is best applied within a branch-and-bound algorithm after a complete set of child-nodes has been generated in the branching stage of the algorithm. For every child-node's partial sequence, each family is 'tested', and a node will be eliminated if one or more families fail to satisfy (1). It can be observed that application of this rule allows the early detection of sequences that would later be eliminated by Rule 4. Computational studies shows the rule to be effective; its tree-pruning advantages clearly outweigh the $O(G)$ computational expense per node (see Reference [19]). The reader is referred to References [6, 13] for discussion of the previously proposed dominance rules.

The *dynamic programming dominance rule* (DPDR) proposed by Crauwels *et al.* can be implemented in the same manner. In our implementation it is deployed after Rule 6, in order to minimize the number of nodes investigated; the DPDR is demanding both of CPU time and memory resources. The effect of the DPDR on the computation time of our branch-and-bound algorithm is studied in Section 8.

4. A DISTRIBUTIVE LOWER BOUND

In this and the following section we introduce new lower bounds for the $1|s_i|\sum wC$ problem which are suitable for use with instances with non-negative integer processing and set-up times distributed over an unrestricted range. The running time properties of the lower bounds are such that they can be calculated in $O(N)$ time at the nodes of a branch-and-bound algorithm. In practice, they can be computed more efficiently than the Lagrangian lower bound of Crauwels *et al.*, and compared to the lower bound of Mason and Anderson they achieve significantly greater proximity to the optimal flowtime.

The first of the new lower bounds *distributes* the set-up time for each family to the jobs of this family, and then considers all family set-up times to be zero. This transforms the original instance

I of the $1|s_i|\sum wC$ problem to an instance I' of the $1||\sum wC$ problem. It is well known that the latter problem can be solved optimally (in polynomial time) by arranging jobs in SWPT order [26]. The transformation from I to I' modifies the processing time of each job $a_{i[j]}$ $(1 \leqslant i \leqslant G,\ 1 \leqslant j \leqslant N_i)$ according to

$$p'_{i[j]} = p_{i[j]} + \beta_{i[j]} s_i \tag{2}$$

where $0 \leqslant \beta_{i[j]} \leqslant 1$ and $\sum_{j=1}^{N_i} \beta_{i[j]} \leqslant 1$. All other data is identical for both instances, except for the family set-up times which are removed from I'.

The validity of such a lower bound is simple to establish. Consider a sequence S of the jobs in instance I. Distribution of set-up times to create instance I', and application of the same sequence S, cannot lead to an increase in the contribution of (distributed) set-up time to the completion time of any job. Thus $C_{i[j]}(I) \geqslant C_{i[j]}(I')$ for all jobs $a_{i[j]}$, and the flowtime of S given instance I' is no greater than the flowtime of S given instance I. From this it follows that by optimally solving instance I' of the $1||\sum wC$ problem we obtain a lower bound for the original instance I with family set-up times. It is also obvious that a more effective lower bound will result when $\sum_{j=1}^{N_i} \beta_{i[j]} = 1$.

To specify a distributive lower bound it is necessary to define a *distribution scheme* governing the values of β. Although a family set-up time may, in fact, be distributed in any fashion amongst the jobs of that family, some distribution schemes will be more effective than others. For the $1|s_i|\sum wC$ problem, there exists a particular scheme which dominates all alternatives. In this dominant distribution scheme, the set-up time for each and every family is distributed with the aim of providing equal weighted processing times for all jobs belonging to the family. If insufficient set-up time is available for some family i, the k jobs receiving distributed set-up time $(k < N_i)$ are those with the least weighted processing time in I, and the set-up time s_i is distributed such that the weighted processing times of these 'first' k jobs are identical in the modified instance I'. Proof of this result can be found in Reference [19].

Distribution of set-up times according to the above distribution scheme provides job $a_{i[j]}$ with a 'delay' due to distributed set-up time to its family. This delay is equal to s_i time units if $\beta_{i[j]} = 0$, or $\sum_{j'=1}^{j} \beta_{i[j']} s_i \leqslant s_i$ if $\beta_{i[j]} > 0$. In any feasible sequence for the $1|s_i|\sum wC$ problem every job $a_{i[j]}$ is 'delayed' by at least s_i time units due to scheduled set-ups for its own family.

A correction constant may be added to the value of any distributive lower bound, in order to compensate for the fact that some jobs are not delayed by a full s_i time units due to distributed set-up times for their family. This correction constant takes a value as given by (3)

$$\sum_{i=1}^{G} \sum_{j=1}^{N_i} w_{i[j]} s_i \left(1 - \sum_{j'=1}^{j} \beta_{i[j']}\right) \tag{3}$$

It can be observed that there is a term for each job $a_{i[j]}$ in an expansion of (3). Each term ensures that the respective job has a total delay due to set-ups for its own family i equal to s_i time units.

Taking any job $a_{i[j]}$ in an arbitrary sequence S, the difference between the completion times of this job in instances I and I' cannot be less than the value of the relevant term in (3). Thus, a distributive lower bound with a correction constant will be a valid lower bound for $1|s_i|\sum wC$. The value of (3) may be calculated in $O(N)$ time, and from this point onwards we will assume that a distributive lower bound includes the correction constant.

We note that the distributive lower bound idea is widely applicable in the development of lower bounds for scheduling problems with sequence-independent set-up times. Crauwels [14] investigates such a bound for the $1|s_i|\sum U$ problem, this bound being developed independently from our own work. The lower bound for $1|s_i|L_{\max}$ devised by Hariri and Potts [27] is also similar in some ways to a distributive lower bound. Two further examples will be considered here, one involving parallel machines and the other dynamic job arrivals.

In the $R|s_i|\sum C$ problem the allocation of jobs to machines in an optimal schedule is generally not known in advance. Thus, a maximum of s_i time units may be distributed to family i jobs ($1 \leqslant i \leqslant G$), regardless of the number of machines. After distribution of set-up times, the resulting $R\|\sum C$ problem may be solved in $O(N^3)$ time using the optimal algorithms proposed by Horn [28] or Bruno *et al.* [29]. If the parallel machines are identical the problem without set-up times can be solved in $O(N \log N)$ time [30, 31].

For the $1|r_j, s_i|\sum C$ problem, construction of an alternative instance solvable in polynomial time as a $1|r_j, pmtn|\sum C$ problem (see Reference [30]) follows expression (2) as well as $r'_{i[j]} = r_{i[j]} - s_i$, where $r'_{i[j]}$ is the ready time of job $a_{i[j]}$ in the transformed instance I'. Note that this ready time applies to the entire $p'_{i[j]}$ processing time units of $a_{i[j]}$ in I'. The necessity for the adjustment of job ready times can be established in a straightforward manner, and mirrors the need for due dates to be modified in Crauwels' lower bound for the $1|s_i|\sum U$ problem.

5. A PRECEDENCE-BASED LOWER BOUND

The second of the new bounds is similar to the lower bound devised by Schutten *et al.* [32] for solution of the $1|r_j, s_i|L_{\max}$ problem. Lawler [33] has shown that the $1|\text{series–parallel}|\sum wC$ problem can be solved in polynomial time. In the case of the $1|\text{tree}|\sum wC$ problem, Horn's algorithm [34] can be used to provide an optimal sequence in $O(N \log N)$ time. The 'SWPT-within-families' rule for $1|s_i|\sum wC$ implies that a chain precedence structure can be imposed between the jobs of each family without increasing the flowtime of an optimal sequence or making such a sequence infeasible. Furthermore, a set-up must precede the first job of a family.

The basis of a lower bound for the $1|s_i|\sum wC$ problem is seen in the above. An instance I of $1|s_i|\sum wC$ may be transformed into an instance I' of $1|\text{chain}|\sum wC$, and the optimal flowtime of I' used as a lower bound for I. The transformation is straightforward. First, order (and index) the jobs of I according to the 'SWPT-within-families' rule. Then, for each family, impose precedence constraints such that job $a_{i[j]}$ becomes a direct predecessor of $a_{i[j+1]}$ for all $1 \leqslant j < N_i$, and introduce for each family a dummy set-up job $a_{i[0]}$ of weight zero and processing time s_i. The resulting instance I' can be solved simply. It is convenient to refer to this lower bound as the *chain lower bound.*

The solution of I' will be described here in terms of Horn's algorithm. At any decision point, Horn's algorithm calls for the evaluation of each of the jobs currently available. The job selected for sequencing will be that with the best *family tree* of all available jobs. A family tree of some job (node) k in a precedence graph is defined as any tree, contained within this graph, which is rooted at k. For any family tree S one may calculate the combined weight $W(S) = \sum\{w_j : j \in S\}$ and processing time $P(S) = \sum\{p_j : j \in S\}$ of the jobs in S, where p_j is the processing time of the job at node j. For any node k the family tree with the least value of P/W, i.e. smallest total weighted processing time, may be identified. Such a family tree is termed the maximal family tree of node k. The best family tree is the maximal family tree with the smallest value of P/W out of all maximal family trees rooted at nodes corresponding to currently available jobs.

The properties of I' are such that if there were no set-up times, every maximal family tree will contain the node at its root, plus only those nodes corresponding to jobs with the same weighted processing time. This is because the precedence constraints are agreeable with the weighted processing times of jobs. When set-up times are included, a maximal family tree from a node corresponding to some set-up job $a_{i[0]}$ will contain the nodes for the first g_i jobs of family i, whereas a maximal family tree from a node corresponding to some job $a_{i[j]}$ $(j>0)$ will once again only contain the node at its root and nodes corresponding to jobs (of the same family) with the same weighted processing time.

Interestingly, for any family i, g_i is equal to the minimum number of jobs that must be included in the first batch of family i in order for Rule 1 to be satisfied, and is given by

$$g_i = \begin{cases} N_i & \text{if } \dfrac{s_i + \sum_{j=1}^{N_i} p_{i[j]}}{\sum_{j=1}^{N_i} w_{i[j]}} \geqslant \dfrac{p_{i[N_i]}}{w_{i[N_i]}} \\ \min\limits_{g_{i'}} \left\{ g_i' \colon \dfrac{s_i + \sum_{j=1}^{g_i'} p_{i[j]}}{\sum_{j=1}^{g_i'} w_{i[j]}} < \dfrac{p_{i[g_i'+1]}}{w_{i[g_i'+1]}} \right\} & \text{otherwise} \end{cases}$$

It should be clear from the above that Horn's algorithm will always sequence a set-up job $a_{i[0]}$ for family i and jobs $a_{i[1]}$ to $a_{i[g_i]}$ contiguously.

An efficient implementation of the chain lower bound calculates g_i for each family i and then combines jobs $\{a_{i[0]}, \ldots, a_{i[g_i]}\}$ into a composite job with processing time $s_i + \sum_{j=1}^{g_i} p_{i[j]}$ and weight $\sum_{j=1}^{g_i} w_{i[j]}$. The G composite jobs and $N - \sum_{i=1}^{G} g_i$ remaining jobs can then be arranged into a SWPT-ordered sequence. This sequence will be optimal for the modified instance I' and a lower bound for the original instance I of $1|s_i|\sum wC$.

6. LOWER-BOUND COMPARISON

Consider the construction of a sequence S_{D}, for instance I', generated by the distributive lower bound. Within any family, every job receiving distributed set-up time has the same weighted processing time. Let the set of family i jobs with distributed set-up time be $\mathcal{J}_i^{\mathrm{D}}$. If job $a_{i[j]} \in \mathcal{J}_i^{\mathrm{D}}$ is to be appended next to S_{D}, then this and every other job in $\mathcal{J}_i^{\mathrm{D}}$ has a weighted processing time no greater than any other job of family i, and all jobs in $\mathcal{J}_i^{\mathrm{D}}$ can be appended to S_{D}. Thus, there is an optimal solution to I' which has every job in $\mathcal{J}_i^{\mathrm{D}}$ scheduled contiguously and in accordance with their original SWPT order (i.e. $a_{i[j]}$ precedes $a_{i[j+1]}$), for each family i. Furthermore, the weighted processing time of the batch composed of $\mathcal{J}_i^{\mathrm{D}}$ is equal to the weighted processing time of each job in $\mathcal{J}_i^{\mathrm{D}}$ (including distributed set-up time).

The jobs in set $\mathcal{J}_i^{\mathrm{D}}$ can be combined into a composite job that will precede all other family i jobs in at least one optimal solution for instance I'. An optimal solution for I' can be generated by sequencing composite jobs and the jobs not receiving distributed set-up time in SWPT order.

The jobs in set $\mathcal{J}_i^{\mathrm{D}}$ will be the same jobs which are grouped with the dummy set-up job in the chain lower bound, i.e. $\mathcal{J}_i^{\mathrm{D}} = \{a_{i[1]}, \ldots, a_{i[g_i]}\}$. If composite jobs are formed within the distributive lower bound, the set of jobs sequenced by the chain lower bound will be identical in all respects to the set of jobs sequenced by the distributive lower bound. Moreover, the two lower bounds will provide identical sequences. After these sequences are formed, composite jobs may be disaggregated into their components. Let $C_{i[j]}(S)$ be the completion time of job $a_{i[j]}$ given sequence S, S containing jobs and either dummy set-up jobs or distributed set-up times.

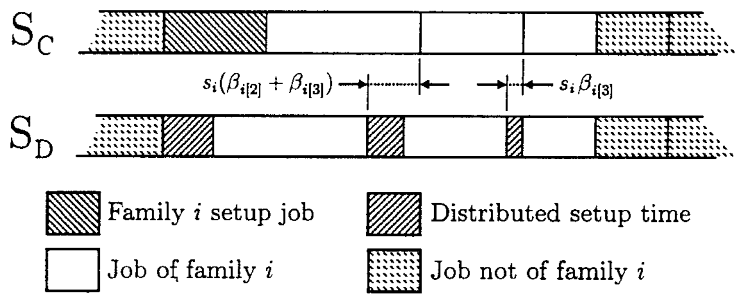

Figure 1. A comparison of completion times generated by the chain and distributive lower bounds.

The difference between the disaggregated sequences for the chain (S_C) and distributive (S_D) lower bounds is highlighted by Figure 1 (in this figure, jobs $a_{i[1]}$ to $a_{i[3]}$ were initially assigned to a single composite job). We see that some completion times in S_D may be less than the corresponding completion times in S_C. It can also be observed that every job $a_{i[j]}$ which has $C_{i[j]}(S_C) > C_{i[j]}(S_D)$ is a job with distributed set-up time in S_D.

The crucial result is that the difference in weighted job completion times

$$w_{i[j]}\,[C_{i[j]}(S_C) - C_{i[j]}(S_D)]$$

for $a_{i[j]}$ takes the same value as the term associated with this job in (3), the correction constant for the distributive lower bound. Thus, the correction constant accounts exactly and equally for the sum of differences in weighted job completion times. Therefore, although the two lower bounds are based upon different ideas, the chain and distributive lower bounds are in fact equivalent.

The Mason and Anderson lower bound is dominated by the chain and distributive lower bounds. This result is established simply. Consider that the flowtime F_W of a sequence is comprised of a component F_W^P due to job processing times and F_W^S due to set-up times. When finding a lower bound for the flowtime of unscheduled jobs, it is noted by Mason and Anderson that the value of F_W^P is minimized when all jobs appear in SWPT order throughout a sequence (regardless of the number or location of set-up times). Define LB_W^P to be this minimum-possible value of F_W^P. For sequence S_C generated by the chain lower bound, the value of F_W^P can be no less than LB_W^P, and typically will be greater than it.

The F_W^S component of S_C is equal to the minimum-possible value LB_W^S credited by the Mason and Anderson lower bound only when in S_C all jobs belonging to either the 'incumbent' family or families with zero set-up times are placed at the beginning of the sequence (in some order), and all other families appear 'unsplit', i.e. as in a 'GT solution'. The families with non-zero set-up times will appear in S_C in non-decreasing order of s/W. If S_C is not of the above form $F_W^S \geqslant \mathrm{LB}_W^S$. Thus due to each component of flowtime being no less in value for the chain lower bound, compared to the Mason and Anderson lower bound, the latter is dominated by the new lower bounds.

7. A BRANCH-AND-BOUND ALGORITHM

Our branch-and-bound algorithm is essentially the same as that of Mason and Anderson, except that we have substituted the more powerful chain lower bound and all of the available dominance rules. The chain lower bound has been chosen over the distributive lower bound as it is easier to implement.

As observed by Crauwels *et al.* [13], pre-processing of an instance can be undertaken in order to reduce the apparent size of the instance. Pre-processing forms composite jobs, these being collections of individual jobs which can be shown to be always sequenced together in at least one optimal sequence for the instance. A composite job has a processing time equal to the sum of the processing times of its component jobs, and its weight is likewise the sum of the weights of its component jobs. The formation of composite jobs merely increases the flowtime of a sequence by a constant, so that the form of an optimal solution to the instance is not altered.

Dominance Rule 1 states that in an optimal solution for a $1|s_i|\sum wC$ problem any batch starting with a given job $a_{i[j]}$ must contain enough jobs to enable the WPT of the next unscheduled job of the same family to be greater than the current WMPT of the batch. Let $\lambda_{\min}(i,j)$ denote the minimum number of jobs in a batch beginning with $a_{i[j]}$. For any job the value of $\lambda_{\min}(i,j)$ can be determined by applying

$$\lambda_{\min}(i,j)=\begin{cases} N_i-j+1 & \text{if } \dfrac{s_i+\sum_{k=j}^{N_i} p_{i[k]}}{\sum_{k=j}^{N_i} w_{i[k]}} \geqslant \dfrac{p_{i[N_i]}}{w_{i[N_i]}} \\ \min\limits_{\lambda'}\left\{\lambda'\colon \dfrac{s_i+\sum_{k=j}^{j+\lambda'-1} p_{i[k]}}{\sum_{k=j}^{j+\lambda'-1} w_{i[k]}} < \dfrac{p_{i[j+\lambda']}}{w_{i[j+\lambda']}}\right\} & \text{otherwise} \end{cases}$$

Crauwels *et al.* determine $\lambda_{\min}(i,1)$ for the first job of each family during pre-processing, and use this information to form composite jobs. Composite jobs of same-family jobs with identical values of weighted processing time are also formed.

In the implementation of the branch-and-bound algorithm described in this paper the value of $\lambda_{\min}(i,j)$ is calculated for every job of every family during pre-processing. For the first job of each family, and for multiple jobs with identical WPT values, composite jobs are formed in the manner of Crauwels *et al.*

For a job $a_{i[j]}$ not included in these composite jobs, $\lambda_{\min}(i,j)$ is recorded along with other data relating to the 'minimum size' of a batch starting with job $a_{i[j]}$. This data consists of the following: a 'reference' to the last job $a_{i[j']}$ which must be included in a batch starting with $a_{i[j]}$; the total time of this 'minimum size' batch starting with $a_{i[j]}$, $T=s_i+\sum_{k=j}^{j'} p_{i[k]}$; the weight of the 'minimum size' batch, $W=\sum_{k=j}^{j'} w_{i[k]}$; and the 'composite job correction constant', $\mathrm{CJC}=\sum_{k=j+1}^{j'}\left(p_{i[k]}\sum_{k'=j}^{k-1} w_{i[k']}\right)$.

The storage of this data allows Rule 1 to be satisfied during a single step in the construction of a new batch. The current schedule completion time t can be updated by adding T, and the value of CJC is such that the value of F_W (for the partial sequence) can be updated using $F_W \leftarrow F_W + \mathrm{Wt} - \mathrm{CJC}$. Clearly, a number of operations may be saved through use of this approach, which can be compared to a job-by-job approach to the satisfaction of Rule 1. Furthermore, the same data is also of use when calculating the lower bound at a node. The data in question is able to be determined in $O(N)$ time during an initial stage of the branch-and-bound algorithm.

An initial upper bound for the algorithm is obtained from a high-quality heuristic solution. In our implementation we have used a modified version of the binary-representation-based descent heuristic developed by Crauwels. The implemented heuristic mimics the operation of the 'binary-based' heuristic, but uses a permutation-based representation in order to reduce the computational burden associated with the encoding and decoding of sequences. This refinement has led to very significant computation time gains in practice.

The number of starts given to the heuristic was varied depending upon the projected difficulty of the instance (i.e. based upon the set-up times and number of jobs). Three starts were used for 30 and 40 job instances, while 10 starts were used when solving 60 and 70 job instances. For instances with 50 jobs, either 3 or 10 starts were used depending on whether the instance was generated with 'medium' or 'small' set-up times (respectively).

The decision to vary the number of starts stemmed from our observation that average branch-and-bound computation time could be reduced by up to a second or more if a high-quality initial upper bound was used when solving difficult instances (e.g. those with 60 or more jobs), compared to a case where the initial upper bound deviates significantly from optimal. This effect was observed by solving a representative set of instances a number of times, on each occasion varying the proximity of the initial upper bound to the optimal solution. In general, we consider a high-quality upper bound to be such that we can be confident it is within at least 0.1 per cent of the optimal solution. Such proximity is quite easily achieved by our chosen heuristic.

The procedure for the lower bound is formulated in such a way as to allow its evaluation in $O(N)$ time at the nodes of a branch-and-bound search tree. This running-time bound is achieved by careful calculation and storage of key elements of data in the initialization stage of the algorithm. Some of this data is the minimum batch size information referred to above. In the most important step two lists are created. In both lists an element is assigned to each job, and the elements are sorted. Elements in the 'jobs' list are ordered according to SWPT for individual jobs without set-up times. In the 'batches' list elements are ordered according to the SWMPT of the 'minimum size' batch beginning with the job concerned. Ties are broken by family index and then by job index. These two lists are merged into one list L which has $2N$ elements arranged in non-decreasing time/weight order of their associated jobs or batches. The number of operations required for the initialization phase is $O(N \log N)$.

When calculating the lower-bound value at a node in the search tree, list L is scanned once. This scan begins with the first element in the list, element L_1, and then progresses through the elements $L_2, \ldots, L_{2N}$ in turn. Consider the case where the current element L_k belonged to the 'batches' list, is assigned to the next unscheduled job $a_{i[j]}$ of some family i, and a set-up for family i is allowed but not yet scheduled in the lower-bound sequence. In this case $a_{i[j]}$ will be sequenced, along with a set-up for family i and any other jobs in the relevant 'minimum size' batch. Alternatively, the job $a_{i[j]}$ to which L_k is assigned will be sequenced if L_k belonged to the 'jobs' list, $a_{i[j]}$ is the next unscheduled job of its family, and no further set-ups for family i are allowed in the lower-bound sequence.

Element L_k is ignored if neither of these two cases holds. Naturally, information about the next unscheduled job of a family is updated whenever jobs are added to the lower-bound sequence. The above procedure allows calculation of the lower bound in $O(N)$ operations.

8. COMPUTATIONAL EXPERIENCE

The performance of the branch-and-bound algorithm has been evaluated using two series of computational experiments. All computational tests have been carried out on an desktop PC equipped with a 133 MHz processor and 32 megabytes of RAM, and the algorithms have been coded in Pascal using Delphi 3.

The first series of tests involved the solution of 37 100-instance data sets. For this series, instances with 'small', $[0, 50]$, and 'medium', $[0, 100]$, set-up times have been generated, as have instances with $[1, 100]$, $[1, 10]$ and $[1, 1000]$ processing time ranges (in the case of the latter two ranges the set-up time limits have been adjusted accordingly). Job weights were sampled from the range $[1, 10]$, except where noted. All of the above values are integers sampled from uniform distributions.

Most instances have been generated with an 'equal' number of jobs per family (i.e. $\lfloor N/G \rfloor$ or $\lceil N/G \rceil$). This is because our preliminary computational testing showed that, on average, such instances are harder to solve than comparable instances with an 'unequal' number of jobs per family. Correlated with this greater difficulty is an increase in the mean number of batches in an optimal sequence, this providing a plausible explanation for the effect.

We have implemented all available dominance rules, including Rule 6 and the DPDR. The state information for the DPDR is held within a tree-structure, each level corresponding to a different variable in the state vector $(n_1, \ldots, n_G; i)$ for the $1|s_i|\sum wC$ problem (this vector defined as in Reference [1]). We have imposed a limit of one million on the number of nodes in this DPDR tree, this corresponding to about 20 Mbytes of stored information. If this limit was reached the enumeration was terminated.

Some instances from four different sets exhibited a requirement for more than a half million DPDR nodes, and 18 instances in total could not be solved within the DPDR node limit. We did not impose a computation-time limit or a limit on the number of nodes in the search tree. A depth-first search is used, so that the algorithm need only store $O(GN)$ nodes at any one instant, and the DPDR becomes the only memory-intensive aspect of the algorithm.

Table I summarizes the branch-and-bound algorithm performance on instances with small set-up times. The node count for this table includes all nodes from which expansion of the search tree is considered, including those from which no non-dominated child nodes are possible. PTR denotes the range over which processing times are uniformly distributed.

It is seen that instances with 50 or fewer jobs can be efficiently solved by the branch-and-bound algorithm, although computation times for some 50-job instances are in excess of one second. It can be observed from Table I that the new algorithm performs better when processing times are sampled from the restrictive $[1, 10]$ range, an effect that further computational testing has confirmed. This appears to be due primarily to a small improvement in the performance of the lower bound. The dependency of search-tree size upon N is clear from Table I, with the average number of nodes increasing by a factor between about two and five for each additional 10 jobs.

A comparison with the ACT results provided in Crauwels *et al.* [13] indicates that the new branch-and-bound algorithm is faster, on average and in direct terms, for all comparable instance specifications other than those with $N \geqslant 60$ and $G = 15$. We claim, therefore, that our algorithm will be competitive with the Crauwels *et al.* algorithm when processing times are distributed over $[1, 10]$, and that it will be superior when processing times are not restricted. For the instances with $N = 70$, $G = 15$ the DPDR node limit for the new algorithm was exceeded 16 times. Due to the frequency with which the DPDR node limit was reached the use of the new algorithm on instances with more than 60 jobs cannot be recommended unless the number of families is restricted or mean set-up times are greater than those studied in Table I.

When branch-and-bound computation times are excessive the adoption of a heuristic solution is a more attractive alternative. Table I provides the observed average computation time required for computing the initial upper bound and the maximum percentage deviation from optimal (MaxPD) of this upper bound. It is clear from this data that Crauwels' binary-descent heuristic is able to

Table I. Branch-and-bound performance: small set-up times.

Small setup times			Computation Time (ms)		Number of nodes		Root node LB	Initial UB		Batches
N	G	PTR	Mean	Max	Mean	Max	Mean(%)	ACT	MaxPD(%)	Mean
30	4	[1,100]	19	40	108	364	96.22	9	0.19	9.8
	8	[1,100]	33	152	176	947	97.04	10	0.00	15.3
	12	[1,100]	28	130	127	704	97.92	10	0.00	18.1
40	4	[1,100]	42	130	307	1285	95.87	12	1.01	11.6
	8	[1,100]	118	815	710	5752	96.33	14	0.08	17.4
	12	[1,100]	110	502	617	2680	97.09	15	0.11	21.5
50	4	[1,100]	119	240	653	1982	95.64	51	0.44	12.0
	6	[1,100]	306	783	1824	5424	95.72	56	0.09	16.6
	8	[1,100]	486	1777	2684	11274	95.85	58	0.14	19.3
	10	[1,100]	689	5556	3468	29145	96.36	62	0.03	22.3
	12	[1,100]	682	3356	3497	19625	96.68	63	0.00	24.1
	15	[1,100]	612	4297	2940	23201	97.17	69	0.01	26.9
50	8	[1,10]	371	1931	1919	9787	96.63	60	0.11	21.8
		[1,10]*	258	1573	1322	8362	96.46	55	0.02	20.4
		[1,100]†	823	8934	4957	55463	96.11	50	0.11	15.9
		[1,100]‡	487	2853	2531	15827	96.03	63	0.02	20.7
60	4	[1,100]	228	522	1526	4653	95.71	63	0.14	13.4
	8	[1,100]	1979	20403	10726	119888	96.64	73	0.03	21.2
	15	[1,100]	4486	40986	20284	171470	96.64	90	0.01	30.0
70	4	[1,100]	514	2350	3737	18940	95.77	77	0.23	14.6
	8	[1,100]	6695	51918	35415	293231	95.64	89	0.07	22.8
	15	[1,100]	17150	64817	71122	283687	96.34	111	0.05	32.4

*Unequal number of jobs per class
†Unit weights
‡Weights [1,100]

quickly deliver solutions of very high quality. This performance is in agreement with data provided by Crauwels for problems with processing times distributed over $[1,10]$. For the task of scheduling more than 50 jobs this heuristic algorithm will often represent the most attractive solution method.

The search tree is larger for our branch-and-bound algorithm in comparison to that of the algorithm of Crauwels *et al.* Naturally, this is due to the use of our new lower bound, which is not as strong as the Lagrangian bound. However the quickly computed 'chain' lower bound appears to be free of the computation-time penalties that would be incurred when using the Lagrangian lower bound to solve problems with 'unrestricted' ranges of processing times.

We have observed in further testing that the minimization of $\bar{F}$ is generally more difficult than the minimization of F_w, for all instances with set-up times $[0,50]$ and below. This occurs despite the clear trend for F_w problems to have more batches in an optimal sequence compared to $\bar{F}$ problems. For ranges of set-up times closer to the range of processing times the opposite trend has been observed.

A plausible reason for this effect does not come to mind readily. However, we have obtained some insight by looking at the ratio of weighted to unweighted flowtime ($F_w/N\bar{F}$). For 'small' set-up times this ratio had a value of 4.62 when averaged over each instance with 50, 60 or 70 jobs, $[1, 100]$ processing time ranges and 'equal' numbers of jobs per family. A comparison of this value to the expected job weight of 5.5 shows that jobs with higher weights are not uniformly spread throughout optimal sequences for F_w, but instead are concentrated near the beginning of the sequence.

Although this result is expected it does confirm that job weights have a strong influence on the composition of an optimal sequence. Certainly, if job weights were spread over an extraordinarily large range the sequence could be constructed easily, by ignoring time and sequencing largest weight first. Nevertheless, when job weights were distributed over a relatively large range, $[1, 100]$, there was only a minor effect upon any measure of interest (see Table I).

For a similar set of instances the value of $F_w/N\bar{F}$ rose to 4.67 and 4.78 when set-up times were distributed over $[0, 100]$ and $[0, 200]$ (respectively), indicating that weights become less influential as set-up times increase. This is intuitively clear, because larger set-up times are expected to have an increased influence on the form of the optimal sequence. When set-up times are 'medium' or greater, F_w problems are generally more difficult to solve compared to $\bar{F}$ problems, at least for the branch-and-bound algorithm. This reversal in the trend observed for 'small' set-ups is thought to be due to a heightened relative influence of the number of batches and the universal trend for the lower bound to perform less effectively for the F_w objective.

Problems with small set-up times are significantly harder to solve than instances with larger set-up time ranges, as a comparison of Table I with the results for 'medium' set-up times (Table II) shows. This observation is in agreement with previously reported computational experiments [6, 13, 14]. For 'medium' set-up times the ACT values drop markedly compared to 'small' set-up times, as does the search tree size. The lower-bound quality improves compared to the 'small' set-up times case, due to a reduction in the number of batches in an optimal sequence (and thus a reduction in the underestimation of the set-up time contribution to flowtime).

The new algorithm appears to be quite well equipped to solve problems with 'medium' set-up times and up to 70 jobs, as except for one instance with $G = 15$ and $N = 70$ all instances were solved within the DPDR node limit. Only a minor effect of the range of processing times is evident in the data. The variation in ACT, tree size, initial upper- and lower-bound performance between the data sets with $[1, 100]$ and $[1, 1000]$ processing time ranges appears to stem mostly from natural sampling variability, as no distinct patterns have emerged. Importantly, the mean number of batches changed only slightly when the maximum processing time was increased from 100 to 1000, even though a change from 10 to 100 yielded a more noticeable effect.

A second series of branch-and-bound algorithm tests has allowed a thorough investigation of the effect of set-up time to be made. In order to demonstrate the effects of set-up time in the clearest possible manner we have assigned the same set-up time s to each family when generating an instance. In total seven levels of set-up time $(1, 2, 5, 10, 50, 100$ and $500)$ and 13 levels of G were studied. For all instances the number of jobs was fixed at 50, the processing time range was held constant at $[1, 100]$ and weights were uniformly sampled from $[1, 10]$.

The effects of set-up time upon algorithm ACT, maximum computation time and the number of batches in an optimal sequence are clearly seen in Figure 2. This figure reinforces the key observation that the set-up times have a very influential effect on the performance of the branch-and-bound algorithm.

Instances with family set-up times near or below 10 time units are evidently the most difficult to solve using the branch-and-bound algorithm. It is surprising that although SWPT will allow the

Table II. Branch-and-bound performance: medium set-up times.

Medium setup times			Computation time (ms)		Number of nodes		Root node LB	Initial UB		Batches
N	G	PTR	Mean	Max	Mean	Max	Mean(%)	ACT	MaxPD(%)	Mean
50	4	[1,100]	62	152	481	1408	95.42	9	0.19	10.6
	6	[1,100]	146	887	1011	6905	95.66	10	0.00	14.1
	8	[1,100]	232	1113	1477	8401	96.17	10	0.00	16.9
	10	[1,100]	239	2171	1375	12923	96.71	12	1.01	19.2
	12	[1,100]	267	2779	1495	17822	97.09	14	0.08	21.5
	15	[1,100]	194	3242	1048	18959	97.66	15	0.11	23.6
50	8	[1,10]	190	1374	1168	8818	96.72	51	0.44	18.5
	10	[1,10]	240	1831	1370	11263	97.02	56	0.09	20.4
50	8	[1,1000]	216	2263	1344	15351	96.21	58	0.14	16.5
	10	[1,1000]	202	1445	1209	10118	96.81	62	0.03	19.1
60	4	[1,100]	170	490	1055	4315	95.20	63	0.00	11.5
	8	[1,100]	969	4802	5369	27908	95.74	69	0.01	19.0
	15	[1,100]	963	20200	4840	111350	97.25	60	0.11	26.0
70	4	[1,100]	293	1469	1961	11876	95.15	55	0.02	12.0
	8	[1,100]	2432	15407	12904	94272	95.46	63	0.02	19.8
	15	[1,100]	4711	45861	20854	197391	96.79	63	0.14	28.4

branch-and-bound algorithm to quickly deal with a problem without set-up times, even instances with $s = 1$ are not easy to solve optimally.

The relationship present in the plot of ACT against set-up time we believe stems from the combination of two competing effects. The number of batches in an optimal sequence increases as the set-up time decreases towards approximately a fifth of the mean processing time. In addition, we expect that the number of non-dominated nodes at each level of the search tree will increase due to a reduction in the strength of many of the dominance rules. This leads to a marked rise in ACT and a similar degradation in other measures of algorithm efficiency. However, as set-up times continue to be reduced their influence recedes. The SWMPT rule becomes increasingly significant, because the optimal sequence tends towards an SWPT ordering of jobs, and although the search tree will contain more levels due to the continual increase in the number of batches, ANN results indicate that the growth in the number of non-dominated nodes is able to be reversed.

There is a strong tendency for instances with $0.1 \leqslant G/N \leqslant 0.4$ to be the hardest to solve, at a given level of s. This observation is in general agreement with previously published experimental data. The pattern of increases in the mean number of batches, with set-up time and number of families, is both well defined and expected. Of some interest is the result that when the set-up time is 500, i.e. approximately 10 times the mean processing time, we can still expect at least one family to have two or more batches in an optimal solution if $G \lesssim 15$ (for $s = 500$ the mean number of batches was observed to be 3.09 and 15.98 for $G = 2$ and 15, respectively).

Figure 2 also allows us to comment on both the benefits of incorporating the DPDR and the increased difficulty associated with solving the mean flowtime problem. It is apparent that when minimizing F_W the memory-intensive DPDR does not generally assist in reducing ACT, whereas it is generally quite beneficial when minimizing $\bar{F}$. We have also observed that when utilizing

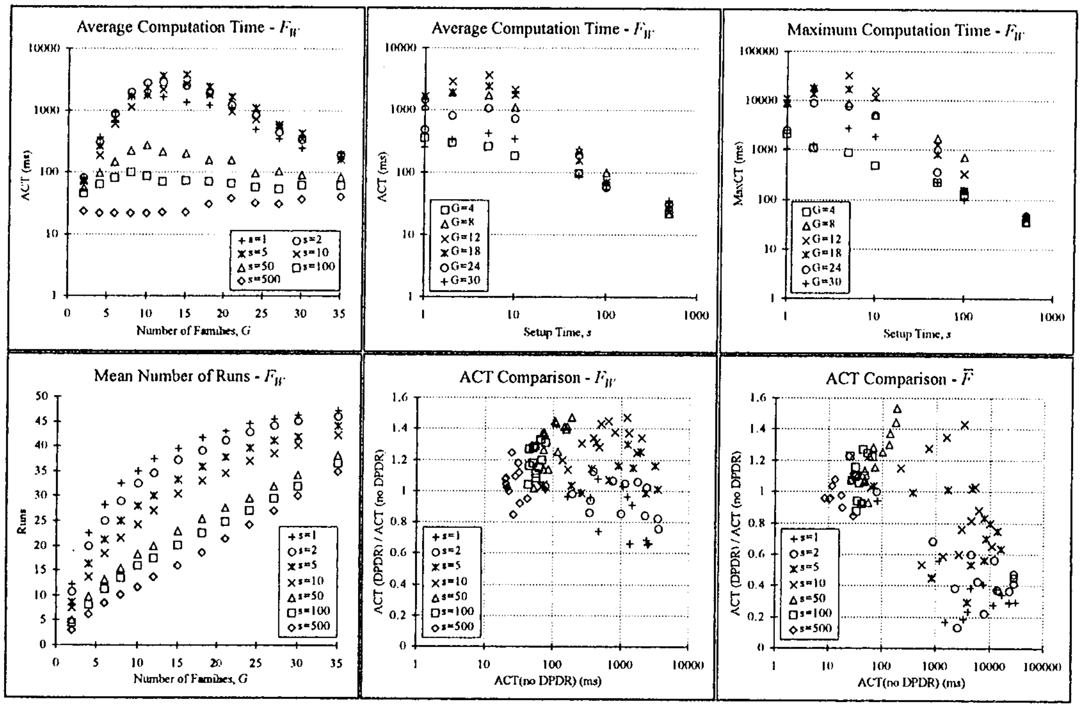

Figure 2. The effect of set-up times on the performance of the branch-and-bound algorithm.

a branch-and-bound algorithm without the DPDR, the average number of nodes increased to 2.3 and 1.3 times the value obtained with the DPDR-equipped algorithm, on average, for $\bar{F}$ and F_W, respectively. When solving instances in order to generate Tables I and II no significant overall difference in either average or maximum computation times was observed between the two versions of the branch-and-bound algorithm.

All of the 9100 instances used in our second series of tests were solved within the DPDR node limit when minimizing F_W, whereas 22 instances were not completely solved for $\bar{F}$ (the greatest number of unsolved instances being four, when $s = 10$ and $G = 18$). From the scale of the plots concerned it is evident that ACT values may be up to 10 times greater when the $\bar{F}$ objective is addressed, in comparison to F_W. We have observed similar trends in DPDR performance and the effect of job weights in maximum computation time data.

9. CONCLUSIONS AND FURTHER WORK

In this paper we have introduced two new lower bounds for the $1|s_i|\sum wC$ problem. These lower bounds have been shown analytically to dominate the lower bound of Mason and Anderson [6]. They have also been shown to be equivalent to each other in terms of lower-bound value. Importantly, the new lower bounds are free of the schedule-length restrictions faced by the Lagrangian lower bound of Crauwels *et al.* [13]. Even though the new lower bounds are not as strong as the Crauwels *et al.* bound, their clear in-practice suitability for use within branch-and-bound algorithms has been highlighted by the computational experiments reported in this paper.

The branch-and-bound algorithm of Mason and Anderson, which was improved in Reference [13], has been developed further within this paper through the addition of a new dominance rule and the substitution of the 'chain' lower bound. Our computational experience with this algorithm has shown it to be able to efficiently solve instances with fifty or more jobs, depending on the values of set-up times. The computational experiments have also clearly shown the effects of important instance parameters, such as the number of families and the duration of set-up times, upon the performance of our algorithm and the composition of optimal sequences for the problem.

We believe that the success of the new branch-and-bound algorithm clearly lies with the efficiency and effectiveness of the lower bound used. This result is of significant importance for the task of scheduling with set-up times, because both of the new lower bounds are based on concepts which can be applied to other problems with sequence-independent set-up times. Some of these potential applications have been noted here, and the development of algorithms for these and other problems represents worthy new directions for future research.

REFERENCES

1. Ghosh JB. Batch scheduling to minimize total completion time. *Operations Research Letters* 1994; **16**:271–275.
2. Liaee MM, Emmons H. Scheduling families of jobs with setup times. *International Journal of Production Economics* 1997; **51**:165–176.
3. Potts CN, Van Wassenhove LN. Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of Operational Research Society* 1992; **43**:395–406.
4. Webster S, Baker KR. Scheduling groups of jobs on a single machine. *Operations Research* 1995; **43**:692–703.
5. Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB. Sequencing and scheduling: algorithms and complexity. In *Handbooks in Operations Research and Management Science*, vol. 4. Elsevier: Amsterdam, 1993.
6. Mason AJ, Anderson EJ. Minimizing flow time on a single machine with job classes and setup times. *Naval Research Logistics* 1991; **38**:333–350.
7. Monma CL, Potts CN. On the complexity of scheduling with batch setup times. *Operations Research* 1989; **37**: 798–804.
8. Rinnooy Kan AHG. *Machine Scheduling Problems*: *Classification*, *Complexity and Computations*. Nijhoff: The Hague, 1976.
9. Potts CN. Scheduling two job classes on a single machine. *Computers and Operations Research* 1991; **18**:411–415.
10. Psaraftis HN. A dynamic programming approach for sequencing groups of identical jobs. *Operations Research* 1980; **28**:1347–1359.
11. Ahn BH, Hyun JH. Single facility multi-class job scheduling. *Computers and Operations Research* 1990; **17**:265–272
12. Bruno J, Sethi R. Task sequencing in a batch environment with setup times. *Foundations of Control Engineering* 1978; **3**:105–107.
13. Crauwels HAJ, Hariri AMA, Potts CN, Van Wassenhove LN. Branch and bound algorithms for single machine scheduling with batch setup times to minimize total weighted completion time. *Annals of Operations Research* 1998; **83**:59–76.
14. Crauwels HAJ. A comparative study of local search methods for one-machine sequencing problems. *Ph.D. Thesis*, De Nayer Institute, Belgium, 1998.
15. Gupta JND. Single facility scheduling with multiple job classes. *European Journal of Operational Research* 1988; **8**:42–45.
16. Crauwels HAJ, Potts CN, Van Wassenhove LN. Local search heuristics for single machine scheduling with batch setup times to minimize total weighted completion time. *Annals of Operation Research* 1997; **70**:261–279.
17. Mason AJ. Genetic algorithms and scheduling problems. *Ph.D. Thesis*, University of Cambridge, 1992.
18. Baker KR. Solving the weighted completion time problem with batch setups. Working Paper #98-118, Amos Tuck School of Business Administration, Dartmouth College, 1998.
19. Dunstall S. A study of models and algorithms for machine scheduling problems with setup times. *Ph.D. Thesis*, University of Melbourne, Australia, 2000.
20. Williams DN, Wirth A. A new heuristic for a single machine scheduling problem with setup times. *Journal of Operational Research Society* 1996; **47**:175–180.
21. Nowicki E, Zdrzalka S. Single machine scheduling with major and minor setup times: a tabu search approach. *Journal of Operational Research Society* 1996; **47**:1054–1064.

22. Liao C-J, Liao L-M. Single facility scheduling with major and minor setups. *Computers and Operations Research* 1997; **24**:169–178.
23. Burbidge JL. *Production Flow Analysis for Planning Group Technology.* Oxford University Press: Oxford, 1989.
24. Sule DR. Sequencing n jobs on two machines with setup, processing and removal times separated. *Naval Research Logistics Quarterly* 1982; **29**:517–519.
25. Dunstall S, Wirth A. Models and algorithms for machine scheduling with setup times. In: *Computer Aided and Integrated Manufacturing Systems*: *Techniques and Applications*, Leondes C (ed.). Gordon & Breach: London, to appear.
26. Smith WE. Various optimizers for single stage production. *Naval Research Logistics Quarterly* 1956; **3**:59–66.
27. Hariri AMA, Potts CN. Single machine scheduling with batch set-up times to minimize maximum lateness. *Annals of Operations Research* 1997; **70**:75–92.
28. Horn WA. Minimizing average flow time with parallel machines. *Operations Research* 1973; **21**:846–847.
29. Bruno J, Coffman Jr EG, Sethi R. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM* 1974; **17**:382–387.
30. Baker KR. *Introduction to Sequencing and Scheduling*. Wiley: New York, 1974.
31. Conway RW, Maxwell WL, Miller LW. *Theory of Scheduling*. Addison-Wesley: Reading, MA, 1967.
32. Schutten JMJ, Van de Velde SL, Zijm WHM. Single-machine scheduling with release dates, due dates and family setup times. *Management Science* 1996; **42**:1165–1174.
33. Lawler EL. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics* 1978; **2**:75–90.
34. Horn WA. Single-machine job sequencing with treelike precedence ordering and linear delay penalties. *SIAM Journal of Applied Mathematics* 1972; **23**:189–202.