Available online at www.sciencedirect.com
ScienceDirect
Computers & Industrial Engineering 52 (2007) 57–70
computers & industrial engineering
www.elsevier.com/locate/dsw

Satisfying due-dates in the presence of sequence dependent family setups with a special comedown structure

Mehmet R. Taner [a], Thom J. Hodgson [b], Russell E. King [b,*], Scott R. Schultz [c]

[a] *Department of Industrial Engineering, Bilkent University, Ankara, TR 06800, Turkey*
[b] *Edward P. Fitts Department of Industrial and Systems Engineering, North Carolina State University, Raleigh, NC 27695, USA*
[c] *Department of Mechanical and Industrial Engineering, Mercer University, Macon, GA 31207, USA*

Received 27 March 2006; received in revised form 25 September 2006; accepted 27 October 2006
Available online 22 December 2006

Abstract

This paper addresses a static, *n*-job, single-machine scheduling problem with sequence dependent family setups. The setup matrix follows a special structure where a constant setup is required only if a job from a smaller indexed family is an immediate successor of one from a larger indexed family. The objective is to minimize the maximum lateness ($L_{\max}$). A two-step neighborhood search procedure and an implicit enumeration scheme are proposed. Both procedures exploit the problem structure. The enumeration scheme produces optimum solutions to small and medium sized problems in reasonable computational times, yet it fails to perform efficiently in larger instances. Computational results show that the heuristic procedure is highly effective, and is efficient even for extremely large problems.

Keywords: Scheduling; Maximum lateness; Sequence dependent family setups; Comedown; Heuristics

1. Introduction

An *n*-job, single-machine scheduling problem with sequence dependent family setups is considered. It is assumed that the jobs are released simultaneously. There are N families with n_i jobs in each family $i = 1, 2, \ldots, N$. Each job j has a given due-date d_j and processing time p_j. Setup is required only if a job from a smaller indexed family is an immediate successor of one from a larger indexed family. When required, the setup time between two families is a given constant s. The initial setup of the machine is also given. The objective is to determine a schedule that minimizes the maximum lateness ($L_{\max}$).

The special setup structure in this problem may be seen in process industries. In Conway, Maxwell, and Miller (1967), it is described as the *comedown problem*, which occurs in the rolling of steel strips. The rollers are slightly scored by the edges of the strip being rolled. Consequently, the next strip in the sequence must be narrower or the rollers must be reground. Another closely related problem is dyeing operations in the textiles

* Corresponding author. Tel.: +1 919 515 5186; fax: +1 919 515 1543.
E-mail address: king@eos.ncsu.edu (R.E. King).

0360-8352/$ - see front matter
doi:10.1016/j.cie.2006.10.008

industry. Usually, a relatively minor setup is involved when lighter to darker shades are dyed in progression. However, when a lighter shade is to be dyed after a dark shade, the dye vessel needs to be cleaned incurring a significant setup time.

In Section 2, a brief overview of research on single-machine scheduling problems with setup times and L_{max} performance criterion is presented. An effective two-step neighborhood search procedure is developed to solve the problem in Section 3. Section 4 describes an implicit enumeration procedure that relies on the ideas proposed in Monma and Potts (1989) and exploits the special structure of the setup matrix. This enumeration scheme is used to evaluate the effectiveness of the proposed heuristic mechanism. Computational results are provided in Section 5. Finally, some conclusions are presented in Section 6.

2. Previous research

There are a number of studies on single-machine problems with setup times and the L_{max} performance criterion. Allahverdi, Gupta, and Aldowaisan (1999), and Potts and Kovalyov (2000) give an excellent review of the scheduling literature with setup considerations. According to Potts and Kovalyov (2000) classification scheme, the problem addressed in this paper belongs to the class of models with family setup times and job availability. They define a batch as a maximal set of jobs that are scheduled contiguously on a machine and share a setup. (Note: In the context of our problem, families in a batch will be sequenced in increasing order of the family index.) In the following, a brief discussion of the most relevant literature is presented. The interested reader is referred to Allahverdi et al. (1999) and/or Potts and Kovalyov (2000) for a more elaborate discussion.

Monma and Potts (1989) use a dynamic programming procedure to minimize L_{max} in the presence of family setups. They establish the NP-hardness of the problem with variable number of families even when the setup times are sequence independent. Although their algorithm efficiently solves the problem when the number of batches is fixed, it is mainly of theoretical value as its complexity is exponential in the number of batches. Ghosh and Gupta (1997) propose an improved backward dynamic programming procedure to solve the same problem. The complexity of their procedure is also exponential in the number of batches. To the best of the authors' knowledge the problem has not yet been addressed heuristically.

Schutten, van deValde, and Zijm (1996) develop a branch and bound procedure to solve the problem with non-simultaneous release times and sequence independent family setups. Hariri and Potts (1997) study the problem with sequence independent setups when all jobs are released at time zero and develop a branch and bound procedure that can solve up to 50-job problems. Since exact algorithms fall short of solving practical sized problems in reasonable computational times, several heuristic algorithms are also proposed. Zdrzalka (1991, 1995) develops heuristic methods and establishes worst-case bounds on the solution. Baker (1999) proposes two heuristic neighborhood search methods that improve the solution obtained by an existing heuristic to minimize L_{max} on a single machine with constant, sequence independent family setups. Baker and Magazine (2000) examine an optimal solution procedure exploiting the special structure of this problem. Their procedure compresses the effective problem size by creating composite jobs and accelerating the enumeration procedure using dominance properties and lower bounds.

To the best of the authors' knowledge, this study is the first heuristic attempt for the problem of minimizing L_{max} on a single-machine subject to sequence dependent family setups and job availability. In addition, it also appears to be the first paper to address the problem with due-dates and the particular set up structure described.

3. Approach and methodology

The approach can be explained in three stages. First, a reduction procedure is devised to transform a given instance of the problem to an equivalent, smaller-sized form. Next, characteristics of an optimal solution are determined. Then, a heuristic solution approach is developed based on these characteristics. The following notation is necessary.

F_i: Set of jobs which belong to family i, where $i = 1, 2, \ldots, N$.
F_i^r: Subset of F_i assigned to batch r.
R: Number of batches.

Table 1 shows the changeover matrix for family setups.

3.1. Problem reduction

Hariri and Potts (1997) prove a result that facilitates the reduction of an instance of the single-machine scheduling problem with sequence independent setups and the objective of minimizing the maximum lateness. The following lemma is a direct result of adaptation of their Theorem 3 to our problem.

Lemma 1. *Let j and j+1 be two consecutive jobs in the earliest due-date order of all jobs belonging to family i ($i = 1, 2, \ldots, N$). If jobs j and $j+1$ satisfy the following condition:*

$$d_j \geqslant d_{j+1} - p_{j+1}$$

then there exists an optimal solution in which jobs j and j+1 are scheduled contiguously.

We skip the proof here as it closely mimics Hariri and Potts' original proof. Along the same lines with Hariri and Potts, when the condition of Lemma 1 is satisfied, we replace jobs j and $j+1$ belonging to the same family by a composite job with processing time $p_j + p_{j+1}$ and due-date min $\{d_j + p_{j+1}, d_{j+1}\}$. The composite job is treated in the same way as an original job in the subsequent computations. The reduction process continues until the condition of Lemma 1 is not satisfied for any two jobs in the current set of composite jobs. This reduction procedure requires $O(n\log(n))$ time. Determination of this complexity is discussed in Appendix A.

3.2. Optimality conditions

The optimality conditions depend upon the relative magnitudes of the processing and setup times and due-dates. It is well-known that when the setup times are all zero, processing the jobs in earliest due-date order minimizes $L_{\max}$. When the setup times are significant, the form of the optimal solution can be represented as follows:

$$\{[F_1^1, F_2^1, \ldots, F_N^1], \ldots, [F_1^R, F_2^R, \ldots, F_N^R]\}.$$

Note that some subsets of F_i can be empty. Monma and Potts (1989) (Theorem 1) prove that there is an optimal schedule where jobs in each family are processed in earliest due-date order. It follows from this result that given the subsets F_i^r for each batch $r = 1, 2, \ldots, R$ and for each family $i = 1, 2, \ldots, N$, sequencing the jobs in each subset F_i^r in due-date order minimizes $L_{\max}$.

Without loss assume that initially the machine is set up. A possible solution is represented by a sequence of batches. Jobs that belong to the same family are sequenced in due-date order and, within each batch, families are sequenced in increasing family index order. Fig. 1 shows two consecutive batches. Let a_i and b_i denote the number of jobs in family i in batches r and $r+1$, respectively. Jobs from all families need not be processed in a batch (i.e. a_i and/or b_i can be zero for some $i \in \{1, 2, \ldots, N\}$). Let job h be the last (a_i^{th}) job of family i in batch r and job k be the first job of family i in batch $r+1$. Let T be the starting time of job h, K be the sum of the processing times of all jobs between job h and job k and let L' be largest lateness among jobs processed between job h and job k.

Table 1
Changeover matrix for family setups

From\To	1	2	...	$N-1$	N
1	0	0	...	0	0
2	s	0	...	0	0
.	.	.	...	.	.
.	.	.	...	.	.
.	.	.	...	.	.
$N-1$	s	s	...	0	0
N	s	s	...	s	0

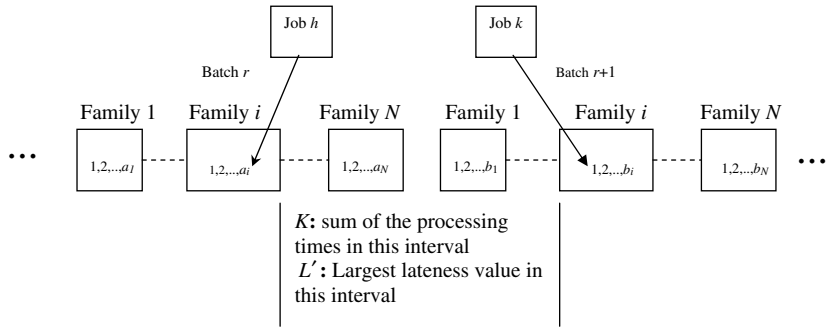

Fig. 1. Two consecutive batches.

Lemma 2. *There exists at least one optimal schedule that satisfies the following three inequalities for each family* $i = 1, 2, \ldots, N$ *and batch* $r = 1, 2, \ldots, R-1$ *such that* $F_i^r \neq \emptyset$ *and* $F_i^{r+1} \neq \emptyset$.

(1) $T + K + s + p_h - d_h \geqslant L'$
(2) $L' + p_k \geqslant T + p_h + K + s + p_k - d_k$
(3) $\max_{j \in F_i^r} \{d_j\} \leqslant \min_{j \in F_i^{r+1}} \{d_j\}$

Proof 1. Suppose that inequality (1) is not satisfied, then job h can be moved to batch $r+1$ with a potential improvement in the value of $L_{\max}$. Similarly, if inequality (2) is not satisfied, job k can be moved to batch r with a potential improvement in the value of $L_{\max}$. Adding p_k to both sides of inequality (1) reveals that any optimum schedule that satisfies inequality (1) may be transformed into a schedule that satisfies inequalities (1) and (2) simultaneously. For such a schedule it must be that

$$T + K + s + p_h + p_k - d_h \geqslant L' + p_k \geqslant T + p_h + K + s + p_k - d_k.$$

Since this implies $d_h \leqslant d_k$, inequality (3) follows directly from (1) and (2). Therefore, any optimum schedule may be modified to satisfy these three inequalities without change in the value of $L_{\max}$. □

3.3. Solution technique

The solution technique has three fundamental schedule improvement techniques.

- *Forward insertion*: The last job of a family within a batch is moved (inserted) before the first job in the corresponding family in the next batch.
- *Backward insertion*: The first job of a family within a batch is moved (inserted) after the last job in the corresponding family in the preceding batch.
- *Setup insertion*: An additional setup is inserted after the job that has lateness equal to the $L_{\max}$ value (referred to as the $L_{\max}$ job here on) so that some jobs with larger due-dates can be delayed until after this new setup to expedite the completion of urgent jobs.

Forward and backward insertion moves are pursued under the conditions that follow from Lemma 2. Define the following additional notation:

f_j: Job j's family index.
I: The job being considered for insertion.
J: The job with the maximum lateness between job I's current and potential insertion points including job I.
J': The job with the maximum lateness between job I's current and potential insertion points not including job I.

K_J: The sum of the processing times of all jobs after job J and before the potential insertion point of job I.
L_j: The lateness of job j.
T^J: The total processing and setup time up to and including job J in the current sequence.

Corollary 1. *Forward insertion of job I (Fig. 2) should be pursued if* $s + K_J < d_I - d_J$ *when* $f_J \geqslant f_I$, *or if* $K_J < d_I - d_J$ *when* $f_J < f_I$.

It is clear that no job other than job I will have a larger lateness after the insertion. Thus, the insertion will result in an improvement if $T^J + K_J + s - d_I < T^J - d_J$ *for* $f_J \geqslant f_I$ *(Note that* T^J *includes job I's processing time). Cancelling like terms on each side and rearranging yields* $s + K_J < d_I - d_J$. *When* $f_J < f_I$, *job J is in the same batch as the insertion point and so* T^J *includes the setup time for that batch as well as job I's processing time. Thus, the insertion will result in an improvement if* $T^J + K_J - d_I < T^J - d_J$. *Cancelling like terms yields* $K_J < d_I - d_J$.

Corollary 2. *Backward insertion of a job I (Fig. 3) should be pursued if* $L_{J'} + p_I < L_J$.

Jobs J and J′ are the same except if J = I. Clearly, this inequality can only be true if job J is job I. In this case, no job other than job I can have a smaller lateness value after the insertion and the backward insertion results in an improvement if the largest of the increased lateness values does not exceed the current lateness of job I (i.e. $L_{J'} + p_I < L_I$).

The following heuristic procedure is based on Lemma 2, and Corollaries 1 and 2.

Algorithm 1

Step 0: Set $r = 1$ and $S_r = \{[F_1^1, F_2^1, \ldots, F_N^1]\}$ where jobs within each family are in earliest due-date order. This is the optimal schedule with a single batch.
Step 1: Insert a setup immediately following the $L_{\max}$ job.
Step 2: Set $r = r + 1$ and $S_r = \{[F_1^1, F_2^1, \ldots, F_N^1], \ldots, [F_1^r, F_2^r, \ldots, F_N^r]\}$.
Step 3: Consider all forward insertion moves according to Corollary 1. If at least one move has been made, repeat Step 3.

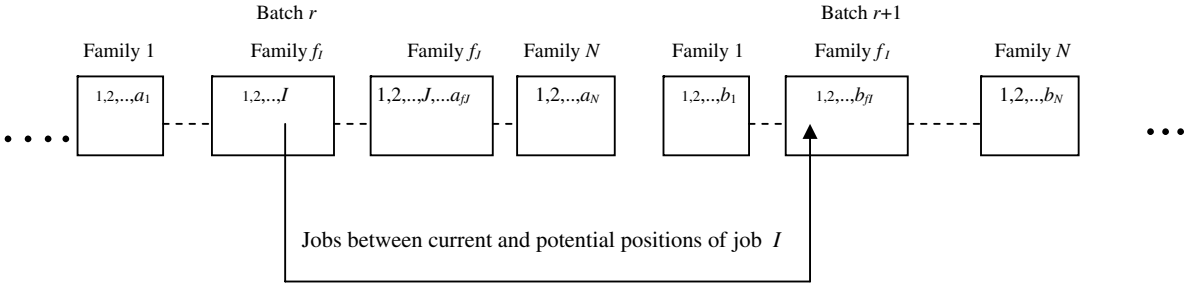

Fig. 2. Forward insertion of job *I*.

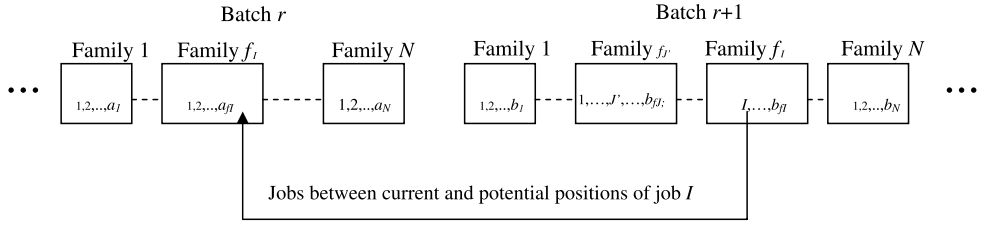

Fig. 3. Backward insertion of job *I*.

Step 4: Consider all backward insertion moves according to Corollary 2. If at least one move has been made, repeat Step 4.
Step 5: If at least one backward insertion was made then repeat at Step 3.
Step 6: Save S_r and its associated maximum lateness value, $L^r_{\max}$. If $L^r_{\max}$ comes from a job that was shifted forward in the schedule due to the most recent setup insertion, go back to S_{r-1} insert a setup immediately following the $L^r_{\max}$ job, replace S_r with the resulting schedule and go to Step 3.
Step 7: If the final setup is followed by a vacuous batch, set $r = r - 1$, report the resulting schedule (i.e., S_{r+1} with the last batch deleted) and the corresponding $L_{\max}$ value and STOP. Otherwise, go to Step 1.

The time complexity of this algorithm is $O(n^3 \log(n))$. Determination of this complexity is discussed in Appendix A.

Example 1. This example illustrates the algorithm on a simple scenario with three families and nine jobs. Consider the problem given in Table 2a. (In the table, the batch number is given with the family identifier, e.g., 1-1 under the Family column represents family 1, batch 1.) Suppose that the initial setup of the machine is for family 1, and $s = 125$. The jobs within each family are already in due-date order. Hence, Table 2a shows the optimum schedule with a single batch. The algorithm terminates in 5 steps as outlined in Tables 2a–2f. The $L_{\max}$ value at each step is shown in a box in the L column. The optimum solution has three batches, and an $L_{\max}$ value of zero.

3.4. A post processing scheme

Initial experimentation with Algorithm 1 indicated a propensity for the procedure to terminate at a local optimum respect to the neighborhood structure. An improvement procedure was proposed as a post processing procedure based on the results of the initial experimentation. This procedure alters the schedule obtained from Algorithm 1 in an attempt to improve it. Forward and backward insertion of jobs and insertion of new setups generate the neighborhood structure used in this procedure. New setups are always inserted at the end of the schedule. Forward and backward insertion moves are pursued as long as they do not cause some job's lateness value to exceed $L_{\max}$ for the current schedule. Propositions 1 and 2, respectively, state the conditions under which a forward or a backward insertion does not make $L_{\max}$ worse and may be pursued in the post processing procedure.

Proposition 1. *Forward insertion of job I (Fig. 2) cannot make L_{max} worse if $T^J + K_J + s - d_I \leqslant L_{max}$ when $f_J \geqslant f_I$, or if $T^J + K_J - d_I \leqslant L_{max}$ when $f_J < f_I$.*

Since no job other than job I will have a larger lateness after the insertion, L_{max} cannot get worse unless the lateness of job I after the insertion exceeds the initial L_{max}. If job I currently precedes the L_{max} job and is inserted after it, the insertion may make the schedule better.

Proposition 2. *Backward insertion of a job (Fig. 3) cannot make L_{max} worse if $L_{J'} + p_I \leqslant L_{max}$.*

Table 2a
Step 0

Family	Job	p	d	L
1-1	1	50	287	−237
1-1	2	79	359	−230
1-1	3	269	1087	−689
2-1	1	4	247	155
2-1	2	30	267	165
2-1	3	167	589	10
3-1	1	3	287	315
3-1	2	86	331	*357*
3-1	3	149	881	−44

Jobs are in due-date order within each family.

Table 2b
Step 1

Family	Job	p	d	L
1-1	1	50	287	−237
1-1	2	79	359	−230
1-1	3	269	1087	−689
2-1	1	4	247	155
2-1	2	30	267	165
2-1	3	167	589	10
3-1	1	3	287	315
3-1	2	86	331	*357*
3-2	3	149	881	81

Insert a setup after job 2 of family 3.

Table 2c
Step 2

Family	Job	p	d	L
1-1	1	50	287	−237
1-1	2	79	359	−230
2-1	1	4	247	−114
2-1	2	30	267	−104
2-1	3	167	589	−259
3-1	1	3	287	46
3-1	2	86	331	*88*
1-2	3	269	1087	−274
3-2	3	149	881	81

Forward insert job 3 of family 1 to batch 2. No other forward insertions possible.

Table 2d
Step 3

Family	Job	p	d	L
1-1	1	50	287	−237
1-1	2	79	359	−230
2-1	1	4	247	−114
2-1	2	30	267	−104
2-1	3	167	589	−259
3-1	1	3	287	46
3-1	2	86	331	*88*
3-1	3	149	881	−313
1-2	3	269	1087	−125

Backward insert job 3 of family 3 to batch 1. No other backward or forward insertions possible.

Note that after the insertion, only job I will have a smaller lateness, the lateness values of all jobs that reside between job I's insertion point and its original position will increase by p_I, and no other job's lateness value will change. Hence, the schedule will not get worse unless $L_{J'} + p_I \leqslant L_{max}$, and it may get better if job I is the L_{max} job.

The following post processing algorithm is proposed based on Propositions 1 and 2.

Algorithm 2

Step 0: Initialize r and $S_r = \{[F_1^1, F_2^1, \ldots, F_N^1], \ldots, [F_1^r, F_2^r, \ldots, F_N^r]\}$ as the number of batches and the schedule in the local optimum solution obtained from Algorithm 1. Set $L'_{\max} = L_{\max}$ and $q = 0$.

Table 2e
Step 4

Family	Job	p	d	L
1-1	1	50	287	−237
1-1	2	79	359	−230
2-1	1	4	247	−114
2-1	2	30	267	−104
2-1	3	167	589	−259
3-1	1	3	287	46
3-1	2	86	331	*88*
3-2	3	149	881	−188
1-3	3	269	1087	0

Insert a setup after job 2 of family 3.

Table 2f
Step 5

Family	Job	p	d	L
1-1	1	50	287	−237
1-1	2	79	359	−230
2-1	1	4	247	−114
2-1	2	30	267	−104
3-1	1	3	287	−121
3-1	2	86	331	−79
2-2	3	167	589	−45
3-2	3	149	881	−188
1-3	3	269	1087	*0*

Forward insert job 3 of family 2. No other forward or backward insertions possible. 1087 is the largest due-date. An additional batch will be vacuous. Stop.

Step 1: Insert a new setup in the end of the schedule. Set $r = r + 1$.
Step 2: Consider all forward insertion moves according to Proposition 1. If at least one move has been made, repeat Step 2.
Step 3: Update the $L_{\max}$ value.
Step 4: Consider all backward insertion moves according to Proposition 2. If at least one move has been made, repeat Step 4.
Step 5: Update $L_{\max}$. If $L_{\max} < L'_{\max}$, set $L'_{\max} = L_{\max}$ and go to Step 2.
Step 6: Consider all forward insertion moves according to Corollary 1. If at least one move is made, repeat Step 6.
Step 7: Consider all backward insertion moves according to Corollary 2. If at least one move has been made, repeat Step 7.
Step 8: Consider all forward insertion moves according to Corollary 1. If at least one move has been made, go to Step 6.
Step 9: Update $L_{\max}$. Set $L'_{\max} = L_{\max}$. If the final batch is not vacuous, go to Step 1.
Step 10: If $q = 0$, set $q = 1$, interchange Steps 2 and 4 and go to Step 2. Otherwise, go to Step 11.
Step 11: Set $r = r - 1$, report $L_{\max}$ and S_r. Stop.

The time complexity of this algorithm is $O(n^4 \log(n))$. Determination of this complexity is discussed in Appendix A.

4. An implicit enumeration procedure

An implicit enumeration procedure was developed in order to determine the effectiveness of the proposed methodology. The procedure relies on the dynamic programming procedure proposed by Monma and Potts (1989) while exploiting the special comedown structure.

The enumeration scheme works in a depth-first fashion. The nodes of the enumeration tree represent jobs, and the level of the node represents the sequence of the job in the schedule. The tree has as many levels as there are jobs. The root node represents the first job in the sequence, and nodes at the bottom level represent the last job in the sequence.

Since an optimal solution in which the jobs belonging to each family are in earliest due-date order exists, the procedure considers only the unscheduled job with the earliest due-date from each family for scheduling next at any level.

At the beginning, the $L_{\max}$ value is set equal to the heuristic solution obtained from Algorithm 2. As this is a depth-first enumeration, a maximum lateness for a feasible job sequence typically is obtained early in the enumeration. When job sequences are evaluated to completion, the incumbent $L_{\max}$ may be updated to reflect a new minimum. As a node is added to the enumeration tree, the lateness of the most recently added job is easily calculated. If the value is greater than or equal to the incumbent $L_{\max}$, then the branch is fathomed.

A branch is also fathomed using a lower bound. The nodes already fixed in the tree represent a partial job sequence. The remaining jobs are unscheduled. To calculate the lower bound, all unscheduled jobs are sequenced in EDD order after the last scheduled job. The completion time for the last scheduled job is known. Therefore, the relaxed completion times (and their lateness values) can then be calculated for the unscheduled jobs. Since the jobs belonging to families with a smaller index than that of the last scheduled job must be preceded by at least one setup, their completion times (and lateness values) are incremented by one setup. To illustrate this, suppose that only two jobs $i \in F_k$ and $j \in F_q$ remain unscheduled at a particular branch where $d_i < d_j$. Let t be the completion time of the most recently scheduled job and l be its family index where $k < l < q$. The lower bound procedure determines the completion times of jobs i and j as $t + s + p_i$ and $t + p_i + p_j$, respectively. When the relaxed completion times (and lateness values) of all unscheduled jobs are determined, the largest lateness value in the relaxed schedule is used as the lower bound for that particular branch. If this lower bound equals or exceeds the incumbent $L_{\max}$, then the branch is fathomed.

The procedure continues until all schedules are fathomed and the minimum $L_{\max}$ value found.

5. Computational results

Algorithms 1 and 2, and the enumeration procedure are implemented in C++. Experimentation is performed on a 1.80 GHz Pentium 4 PC. Test problems are generated with 2, 4, 6, 8, and 10 families (N), and between 8 and 80 jobs (n) distributed evenly to each family. Processing times are generated from the Uniform distribution with a range of [1,19]. Setup times are systematically set equal to 5, 15, and 25. Due-dates are sampled from a Uniform distribution with a lower limit of 1. The range of the distribution is varied as 10, 30, 50, and so on up to the maximum total processing time in all problems except for the largest set in each family. In particular, in the two family 80 job, four family 64 job, six and eight family 48 job, and ten family 40 job problems, this range is varied in increments of 60 as 10, 70, 130, and so on within the same range to keep the total computational time with the enumeration scheme under a reasonable limit. Five randomly generated instances of each problem are attempted using both the heuristic and the enumeration procedure.

In an attempt to compare the performance of our heuristic also with the most relevant attempts found in the literature, Baker's (1999) GAP/CS procedure for the case with sequence independent family setups is adapted and implemented for this problem. However, the adaptation performed poorly as it was originally designed for an inherently different problem. Hence, its results are not reported here.

The results for problems for which the implicit enumeration procedure fails to terminate in 30 min of CPU time are discarded since we then have no basis for comparison of our heuristic algorithm to the optimal solution. Our heuristic computes a schedule in less than a second in all cases (8820 problems). The implicit enumeration procedure completes in less than 30 minutes for about 98 % (8653 problems) of those attempted. About 96% (8332 problems) of the problems for which the optimum schedule could be determined are solved

optimally using Algorithm 1. The post processing mechanism increases this percentage to 99.5% (8611 problems). For all problems not solved by the enumeration algorithm in 30 min CPU time, the incumbent solution had the same value as the heuristic solution when it was terminated.

The results for 2-family problems are shown in Table 3. Problems with 28, 32, 36, 40, and 44 jobs are solved. The heuristic finds the optimal schedule for all (3900) problems in this class. Both procedures run quickly in these problems. The enumeration procedure takes slightly longer CPU time than the heuristic in larger instances with 44 and 80 jobs. Using the heuristic result as an initial upper bound (UB) speeds up the enumeration procedure.

Table 4 displays the results for 4-family problems. Four-family problems with 8, 16, 24, 32, 40, and 64 jobs are attempted. The heuristic gives optimal results for 2106 of the 2118 problems for which the enumeration completes. The twelve unsolved problems have medium due-date ranges, which in general result in longer CPU times with the enumeration scheme. The average and maximum percentage deviations in sub-optimal cases are, respectively, 5.50% and 14.29% in 32-job problems, 4.42% and 7.41% in 40-job problems, and 9.26% and 12.50% in 64-job problems. Both the enumeration and the heuristic run quickly for up to 40-job problems, but the enumeration takes significantly longer for problems with 64 jobs. The enumeration runs faster when the heuristic result is used as an initial upper bound. Not using this first initial upper bound renders three more problems with 64 jobs unsolvable by the enumeration algorithm in the allotted 30-min CPU time.

The results for 6, 8, and 10-family problems are given in Table 5. Six-family problems with 12, 24, 36, and 48 jobs are attempted. The heuristic yields optimum solutions for all 12-job problems. Among those problems for which the optimum solution could be secured, two of the 24-job problems, eight of the 36-job problems, and four of the 48-job problems are solved sub-optimally by the heuristic. The average and maximum percentage deviations from optimal are 2.92% and 4.08%, respectively in the sub-optimal cases with 24 jobs. The average and maximum percentage deviations, respectively, are 4.98% and 9.52% in the sub-optimal cases with 36 jobs, and 3.02% and 5.32% with 48 jobs.

Problems with 16, 32, and 48 jobs and 8 families are attempted. All except one of the 16-job problems are solved optimally by the heuristic. The percentage deviation from optimum in the sub-optimal case is 5.66%. The heuristic gives sub-optimal results for 9 of the 475 32-job problems for which an optimum solution could be obtained. The average and maximum percentage deviations from the optimal, respectively, are 2.26% and 4.17%. Among the 172 48-job problems for which optimum results are found, 168 are solved optimally also by

Table 3
Computational results for 2-family problems

Families	Jobs	s	Problems attempted	# optimal before post processing	# optimal after post processing	Algorithms 1 and 2 avg. CPU (s)	Enumeration avg. CPU (s)	Enumeration without UB avg. CPU (s)
2	28	5	140	139	140	0.00086	0.00007	0.0015
		15	140	138	140	0.00050	0.00014	0.0008
		25	140	138	140	0.00043	0.00007	0.0007
2	32	5	160	156	160	0.00031	0.00063	0.0014
		15	160	154	160	0.00031	0.00050	0.0014
		25	160	157	160	0.00056	0.00051	0.0013
2	36	5	180	178	180	0.00094	0.00050	0.0032
		15	180	178	180	0.00028	0.00011	0.0035
		25	180	173	180	0.00033	0.00028	0.0027
2	40	5	200	193	200	0.00085	0.00050	0.0043
		15	200	194	200	0.00070	0.00030	0.0041
		25	200	193	200	0.00035	0.00050	0.0032
2	44	5	220	214	220	0.00105	0.00087	0.0077
		15	220	212	220	0.00064	0.00064	0.0067
		25	220	214	220	0.00100	0.00032	0.0061
2	80	5	400	388	400	0.00318	0.11768	0.2880
		15	400	374	400	0.00225	0.08585	0.3180
		25	400	363	400	0.00150	0.04970	0.3148

Table 4
Computational results with 4-family problems

Families	Jobs	s	Problems attempted	# optimal before post processing	# optimal after post processing	Algorithms 1 and 2 avg. CPU (s)	Enumeration avg. CPU (s)	Enumeration without UB avg. CPU (s)
4	8	5	40	40	40	0.00075	0.00000	0.0003
		15	40	40	40	0.00000	0.00000	0.0000
		25	40	40	40	0.00025	0.00000	0.0003
4	16	5	80	79	80	0.00038	0.00050	0.0006
		15	80	80	80	0.00025	0.00013	0.0005
		25	80	80	80	0.00025	0.00013	0.0003
4	24	5	120	119	120	0.00142	0.00708	0.0095
		15	120	118	120	0.00092	0.00308	0.0054
		25	120	117	120	0.00067	0.00260	0.0034
4	32	5	160	156	159	0.00194	0.21429	0.3581
		15	160	154	159	0.00156	0.08488	0.1071
		25	160	152	158	0.00050	0.02445	0.0362
4	40	5	200	192	199	0.00381	2.21796	2.6627
		15	200	194	200	0.00221	0.41671	0.5466
		25	200	189	199	0.00130	0.13880	0.1809
4	64	5	99/110[a]	97	98	0.01275	269.1275	307.9050
		15	109/110	97	106	0.00718	62.66461	65.1978
		25	110	96	108	0.00427	21.38174	46.4473

[a] The enumeration procedure produces optimum results for 99 of 110 attempted problems within the 30-min CPU time limit.

Table 5
Computational results with 6, 8 and 10-family problems

Families	Jobs	s	Problems attempted	Optimum before post processing	Optimum after post processing	Algorithms 1 and 2 avg. CPU (s)	Enumeration avg. CPU (s)	Enumeration without UB avg. CPU (s)
6	12	5	60	60	60	0.00050	0.00033	0.0007
		15	60	60	60	0.00050	0.00000	0.0005
		25	60	60	60	0.00050	0.00017	0.0002
6	24	5	120	119	119	0.00218	0.29459	0.3185
		15	120	113	120	0.00117	0.11191	0.1344
		25	120	117	119	0.00083	0.04741	0.0616
6	36	5	178/180	172	176	0.00491	57.12302	72.3744
		15	180	174	180	0.00306	4.85639	8.5275
		25	180	167	174	0.00239	1.90626	2.3971
6	48	5	63/80	60	63	0.00764	435.93093	467.6206
		15	70/80	67	68	0.00363	287.54796	356.5465
		25	77/80	71	75	0.00339	167.95275	191.7284
8	16	5	80	80	80	0.00075	0.04169	0.0509
		15	80	80	80	0.00163	0.00751	0.0089
		25	80	78	79	0.00050	0.00400	0.0053
8	32	5	155/160	149	154	0.00451	155.75249	200.5427
		15	160	153	158	0.00263	28.00793	31.0732
		25	160	149	154	0.00207	5.03504	5.9762
8	48	5	51/80	48	49	0.00928	722.15823	771.3054
		15	57/80	54	55	0.00513	573.22731	619.1070
		25	64/80	58	64	0.00363	424.75965	499.7161
10	20	5	100	99	100	0.00150	3.04555	3.1084
		15	100	96	100	0.00120	0.18190	0.1945
		25	100	97	99	0.00090	0.06846	0.0717
10	40	5	48/70	46	47	0.00914	586.82480	644.0883
		15	52/70	50	52	0.00516	525.74956	597.2685
		25	60/70	59	60	0.00414	362.94666	425.7549

the heuristic. In the sub-optimal cases the average and maximum deviations from the optimum are 2.16% and 3.33%, respectively.

In addition, 20- and 40-job problems with 10 families are attempted. All but one of the 20-job problems are solved optimally using the heuristic. The percentage deviation from the optimum in the sub-optimal case is 12.28%. Among the 210 attempted problems with 40 jobs, the enumeration algorithm solves 160 within a 30-min CPU time. The heuristic algorithm gives a sub-optimal result in only one of these problems. The percentage deviation from the optimum is 13.89% in this problem.

In general, computational times for the heuristic are less than the enumeration scheme. The difference becomes more significant as the number of jobs is increased. Providing the heuristic result as an initial upper bound increases the speed of the enumeration scheme. Not using this initial bound renders thirty-two more problems in this set unsolvable by the enumeration algorithm within the 30-min CPU time limit.

Finally, 1000 randomly generated, 6-family, 24-job problems with a due-date range of 210 and setup time of 5 are evaluated. This particular scenario is chosen since one of the five randomly generated problems with these parameters is previously solved sub-optimally by the heuristic. Of these, 989 (~99%) are solved optimally by the heuristic. The average and maximum percentage deviations from optimum $L_{\max}$ in the 11 cases not solved optimally are 3.95% and 14.28%, respectively. It should be noted that the worst case (14.28%) deviation corresponds to less than one-third of the average job processing time. The average CPU time for the heuristic (0.00098-s) is about four orders of magnitude smaller than that of the enumeration scheme (2.15-s).

Additional experimentation is performed to observe the practical complexity of the heuristic procedure. First, a set of problems with 1500 jobs, a due-date range of 15,000 and a setup time of 15 are considered. The number of families is varied between 2 and 50 in increments of 2. Five randomly generated instances of each problem are solved. Fig. 4 shows the average CPU time in seconds. CPU time seems to vary as an approximate linear function of the number of families. Next, the number of jobs is varied between 12 and 4800 in increments of 12 for 6-family problems with a setup time of 15. The due-date range is set equal to 10 times the number of jobs. Five random instances of each problem are solved. Fig. 5 shows the average CPU time. CPU time seems to vary as a polynomial function of the number of jobs.

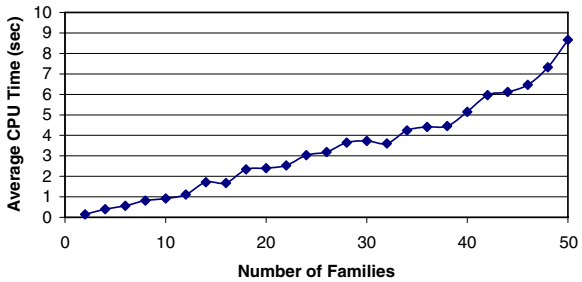

Fig. 4. CPU time as a function of the number of families.

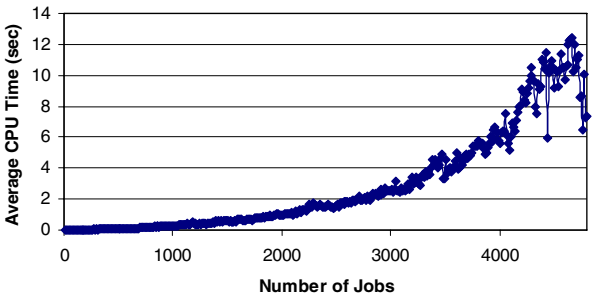

Fig. 5. CPU time as a function of the number of jobs.

6. Concluding remarks

In this paper an n-job, static single-machine scheduling problem with sequence dependent family setups is addressed. Consistent with the textile dyeing operations, setup is required only if a job from a smaller indexed family (lighter colored fabric) is an immediate successor of one from a bigger indexed family (darker colored fabric). An exact implicit enumeration scheme and an effective two-step neighborhood search procedure are developed to minimize the maximum lateness. The exact procedure solves small and medium sized instances efficiently, however, its computational requirements are formidable in large instances. Extensive computational experimentation with the proposed heuristic procedure justifies that the technique is highly effective, and is computationally efficient even for large problems.

Acknowledgements

The authors wish to thank the three anonymous referees for their valuable comments on an earlier draft of this paper.

Appendix A

Define

N: Number of families
n: Number of jobs
where $N \leqslant n$.

Initialization

Put all jobs in earliest due-date order. Complexity: $O(n\log(n))$.

Reduction procedure

Complexity associated with checking if any of the n jobs can be combined with the next job (e.g., if all jobs were in the same family) is $O(\log(n))$. Even if two jobs were to be combined at a time, the procedure would be repeated no more than n times. Thus the overall complexity is $O(n\log(n))$.

Algorithm 1

Complexity associated with checking if any forward (backward) insertion moves are possible in a single pass is $\log(n)$. For a given number of batches, even if each job were to be moved to the next (previous) batch in the sequence one at a time, the maximum number of self-calls for the forward (backward) insertion procedure would be n. In addition, since each backward insertion move may result in a revocation of step 3, there are a maximum of n recursions of steps 3–8 as a result of a backward insertion move. Finally, the maximum number of batches is limited by n (i.e., each job forming a separate batch). Hence, the complexity of Algorithm 1 is $O(n^3\log(n))$.

Algorithm 2

Complexity associated with checking if any forward insertion moves can be made at a single pass in step 2 is $\log(n)$. Even if only one job were to be moved in each pass, the maximum number of self-calls would be n. These observations are also applicable to step 4. The maximum number of revocations of step 2 as a result of the test performed in step 5 is n^2, that is, the number of combinations of all possible forward and backward insertion moves, respectively in steps 2 and 4 for a given number of batches. The complexity of the process that takes place in steps 6-8 is the same as the complexity of Algorithm 1 for a given number of batches (i.e.,

$O(n^2\log(n)))$. Once again, the maximum number of batches is limited by n. Hence, the complexity of Algorithm 2 is $O(n^4\log(n))$.

Based on these observations the overall complexity of the proposed heuristic procedure is $O(n^4\log(n))$.

References

Allahverdi, A., Gupta, J. N. D., & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega, The International Journal of Management Science, 27*, 219–239.

Baker, K. R. (1999). Heuristic procedures for scheduling job families with setups and due-dates. *Naval Research Logistics, 46*, 978–991.

Baker, K. R., & Magazine, M. J. (2000). Minimizing maximum lateness with job families. *European Journal of Operational Research, 127*, 126–139.

Conway, R. W., Maxwell, W. L., & Miller, L. W. (1967). *Theory of scheduling*. Reading, MA: Addison Wesley Publishing Co, pp. 54.

Ghosh, J. B., & Gupta, J. N. D. (1997). Batch scheduling to minimize maximum lateness. *Operations Research Letters, 21*, 77–80.

Hariri, A. M. A., & Potts, C. N. (1997). Single machine scheduling with batch setup times to minimize maximum lateness. *Annals of Operations Research, 70*, 75–92.

Monma, C. L., & Potts, C. N. (1989). On the complexity of scheduling with batch setups. *Operations Research, 37*, 798–804.

Potts, C. N., & Kovalyov, M. Y. (2000). Scheduling with batching: a review. *European Journal of Operational Research, 120*, 228–249.

Schutten, J. M. J., van deValde, S. L., & Zijm, W. H. M. (1996). Single machine scheduling with release dates, due-dates and family setup times. *Management Science, 42*, 1165–1174.

Zdrzalka, S. (1991). Approximation algorithms for single machine sequencing with delivery times and unit batch set-up times. *European Journal of Operational Research, 51*, 199–209.

Zdrzalka, S. (1995). Analysis of approximation algorithms for single machine scheduling with delivery times and sequence independent batch setup times. *European Journal of Operational Research, 80*, 371–380.