International Journal of Production Research

ISSN: 0020-7543 (Print) 1366-588X (Online) Journal homepage: http://www.tandfonline.com/loi/tprs20

Minimizing L_{max} for the single machine scheduling problem with family set-ups

S. R. Schultz , T. J. Hodgson , R. E. King & M. R. Taner

To cite this article: S. R. Schultz , T. J. Hodgson , R. E. King & M. R. Taner (2004) Minimizing L_{max} for the single machine scheduling problem with family set-ups, International Journal of Production Research, 42:20, 4315-4330, DOI: 10.1080/00207540410001716561

To link to this article: http://dx.doi.org/10.1080/00207540410001716561

Published online: 21 Feb 2007.

INT. J. PROD. RES., 15 OCTOBER 2004,
VOL. 42, NO. 20, 4315–4330

Taylor & Francis
Taylor & Francis Group

Minimizing $L_{\max}$ for the single machine scheduling problem with family set-ups

S. R. SCHULTZ†, T. J. HODGSON‡*, R. E. KING‡ and M. R. TANER§

A procedure for the single machine-scheduling problem of minimizing the maximum lateness for jobs with sequence independent set-ups is presented. The procedure provides optimal/near-optimal solutions over a wide range of problems. It performs well compared with other heuristics, and it is effective in finding solutions for large problems.

1. Introduction

Consider the problem of minimizing the maximum lateness ($L_{\max}$) for a set of jobs to be processed on a single machine. Each job belongs to a given part family. Jobs are denoted with an (i, j) subscript indicating the jth job from family i. Thus, p_{ij} and d_{ij} are the processing time and due date for job (i, j), respectively. The number of jobs in a family i is N_i. When starting the processing of a job with a family different from that of the preceding job, a set-up of length s_i occurs for family i. The set-up time for the new family is independent of the prior family. All jobs are ready to be scheduled at time 0. Lateness (L_{ij}) is defined as the job completion time (C_{ij}) minus the job due date (d_{ij}), i.e. $L_{ij} = C_{ij} - d_{ij}$. This problem in standard scheduling notation is $1/s_{i,b}/L_{\max}$.

The problem was first studied by Bruno and Downey (1978), who showed that the problem of determining whether a schedule exists with no tardy jobs is NP-complete. Monma and Potts (1989) showed that maximum lateness and the number of tardy jobs problems are NP-hard. Schutten *et al.* (1996) developed a branch-and-bound algorithm to solve the problem with non-zero ready times. Hariri and Potts (1997) analysed several heuristics and developed a branch-and-bound algorithm for the case when all jobs are ready at time 0. Computational results in Hariri and Potts (1997) and Schutten *et al.* (1996) indicate difficulty in solving problems with more than 50 jobs. Baker and Magazine (2000) also used a branch-and-bound approach and established that the size of the problems that can be solved is a function of the number of families, the number of jobs per families, the relative size of the set-up time and the relative due date range. For the most difficult categories, they solved problems with up to 60 jobs.

Revision received December 2003.

† Department of Mechanical and Industrial Engineering, Mercer University, Macon, GA 31207, USA.

‡ Department of Industrial Engineering, PO Box 7906, North Carolina State University, Raleigh, NC 27695-7906, USA.

§ Department of Industrial Engineering, Bilkent University, TR-06800 Bilkent, Ankara, Turkey.

* To whom correspondence should be addressed. e-mail: hodgson@eos.ncsu.edu

International Journal of Production Research ISSN 0020–7543 print/ISSN 1366–588X online
http://www.tandf.co.uk/journals
DOI: 10.1080/00207540410001716561

Baker (1999) examined heuristic solution procedures for the problem. An experimental framework based on due date ranges and set-up time factors was developed that identified problem instances difficult to solve. Computational results indicate that his hybrid heuristic (on average) produces results where the difference between the heuristic maximum lateness and optimal maximum lateness is approximately the average job processing time.

In the present paper, a new neighbourhood search heuristic is presented. The procedure is shown to be effective, producing optimal/near-optimal solutions over a wide range of problem instances and is computationally efficient for large problems. It was developed based on properties and theorems presented by Hariri and Potts (1997) and Baker (1999). Of particular interest is Hariri and Potts' problem reduction procedure that identifies conditions under which two jobs from the same family must be scheduled contiguously and can thus be replaced by a single composite job, therefore reducing the overall problem size. Also of interest is Baker's Property 3, which states that there exists an optimal schedule such that the batches are sequenced in non-decreasing order of their batch due dates.

2. Sequencing improvement routine for a given batch partition

2.1. *Optimality conditions for a batch partition*

Conditions for optimality are presented below for a constrained version of the original problem where the number of batches for each part family is fixed. As will be shown, a heuristic is developed that solves a succession of these constrained problems in order to construct a solution to the original problem.

The following definitions are helpful in presenting the optimality conditions.

- Batch — set of successive jobs from the same part family,
- Batch partition $(n_1, n_2, n_3, \ldots)$ — representation of the problem space where n_i is the number of batches of family i. For example, given four part families a batch partition of $(2, 3, 2, 3)$ indicates that the sequence contains two batches for part families 1 and 3, and three batches for part families 2 and 4,
- Neighbouring partition — batch partition created by inserting a new batch into a given partition. The neighbour may incur an increase of one or two batches depending on how the new batch is inserted,
- Subset of jobs — within a sequence of jobs, the set of all jobs between and including those from two consecutive batches of a given part family. For example, given a sequence of jobs $(\ldots(i,n) \ldots (k,l) \ldots (i,n+1)\ldots)$, a subset is formed from the sequence of jobs beginning with the batch that ends with job (i,n) through the batch that begins with job $(i, n+1)$.

Given these definitions, there exists at least one optimal schedule that satisfies the following two conditions.

1. Jobs within a family are sequenced in earliest due-date order. There exists an optimal sequence such that jobs within each family are sorted by non-decreasing order of their due dates. Proof of this condition is given by Monma and Potts (1989) and follows the well-known pair-wise switching proof for the $1//L_{\max}$ problem. Henceforth, assume that jobs of a given family i are numbered in increasing due date order, i.e. $d_{ij} \leq d_{ik} \forall j < k$.
2. Batches are sequenced by their batch-adjusted due date. Having defined a batch as consisting of only jobs from the same part family and using

condition 1 above, all jobs within a batch are sequenced in earliest due-date order. Then a due date, d_b, can be assigned to batch b consisting of jobs from family i, such that $d_b = \min_{j \in b}\{d_{ij} + \sum_{k>j} p_{ik}\}$. In other words, the batch-adjusted due date for batch b is the minimum of the due date for some job j plus the sum of all downstream processing in its batch. Using these batch-adjusted due dates, there exists an optimal sequence such that all batches are sorted by non-decreasing order of their due dates. This condition was first noted by Webster and Baker (1995) as a corollary to a property proved earlier in Unal and Kirnan (1992) and Potts and VanWassenhove (1992).

In addition, when constrained by a fixed batch partition, an observation is identified under which an optimal schedule is obtained.

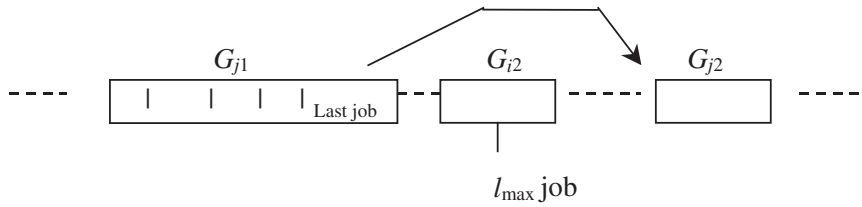
3. All possible exchanges of jobs between batches of a family have been tested. Given a batch partition sorted by optimality condition 2 (batches sorted by batch-adjusted due date), there may exist jobs that can be moved between batches of a like family such that $L_{\max}$ is improved without changing the batch partition.

2.2. *Job moves to improve the sequence*

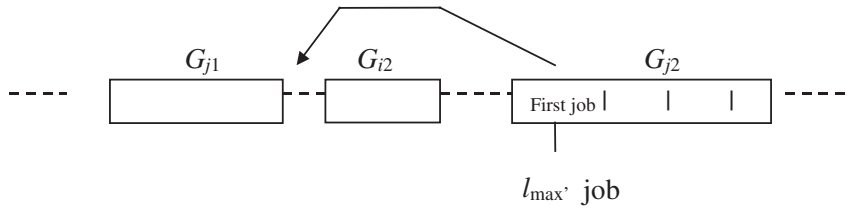
In keeping with observation 3, we consider three types of job moves: right job move, left job move and left $L_{\max}$ move. For the first two, consider a subset of jobs defined by a given part family. Let the $l_{\max}$ job be the one with the largest lateness amongst all jobs in the given subset excluding those of the family that define the subset (first and last batch), and let the $l_{\max}$ batch be the batch to which the $l_{\max}$ job belongs. ($l_{\max}$ is used rather than $L_{\max}$ since the job with the largest lateness in the subset may not be the job with the largest lateness for the entire sequence.)

We make the following observations.

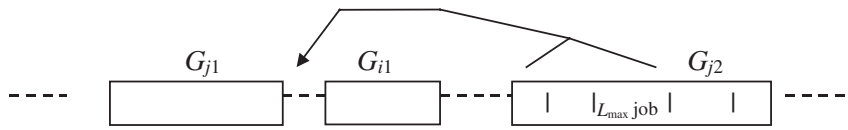
- Consider some given subset. While maintaining the same batch partition, an improvement in the subset's $l_{\max}$ will result when the last job in the first batch in the subset is moved to the first position in the last batch in the subset. With such a move, the lateness of all jobs between the first and last batch will improve, while the lateness of the moved job will increase. However, the lateness of any other job in the first and last batch is unaffected by the move. If the new lateness of the job being moved is less than the original $l_{\max}$, then the move is immediately accepted. Otherwise, with the job moved, all batches in the entire sequence are sorted to satisfy optimality condition 2. If the overall $L_{\max}$ improves, the move is accepted. We define this type of move as a right job move.
- Consider the case where the lateness of the first job of the last batch that defines the subset, call this $l_{\max'}$, is greater than the $l_{\max}$ for the subset. While maintaining the same batch partition, an improvement in the subset's $l_{\max'}$ may result when the considered job is moved to the last position of the first batch in the subset. If the $l_{\max}$ of the subset after the move is less than the original lateness of the moved job, $l_{\max'}$, then the move is immediately accepted. Otherwise, with the considered job moved, all batches in the entire sequence are sorted to satisfy optimality condition 2. If the overall $L_{\max}$ improves, the move is accepted. We define this type of move as a left job move.
- Consider all jobs in the schedule. An improvement in $L_{\max}$ may result when the jobs from the $L_{\max}$ batch preceding and including the $L_{\max}$ job are moved as a



(a) - Right job move



(b) - Left job move



(c) - Left L_{max} move

Figure 1. Examples of job moves.

group to the batch of the same part family preceding the L_{max} batch. After the jobs are moved, the batches are reordered to satisfy optimality condition 2. If the result is an improvement in L_{max}, the jobs are moved. We define this type of move as a left L_{max} move.

To illustrate these moves, consider the examples in figure 1. Let G_{ib} be the bth batch of part family i. Figures 1(a–c) illustrate a right job move, a left job move and a left L_{max} move, respectively.

2.3. *Scheduling heuristic for a fixed batch partition*

For a fixed batch partition, right job moves, left L_{max} moves, left job moves (observation 3) and sorts by batch-adjusted due dates (optimality condition 2) are performed to find a schedule that 'best' minimizes L_{max}. The term 'best' is used since there is no guarantee that an optimal sequence of jobs is constructed. The moves are executed only if the batch partition integrity is maintained. In other words, a right or left job move is not accepted if the considered job comes from a batch containing a single job. Additionally, a sort by batch-adjusted due date is not accepted if batches from the same family end up being consecutive.

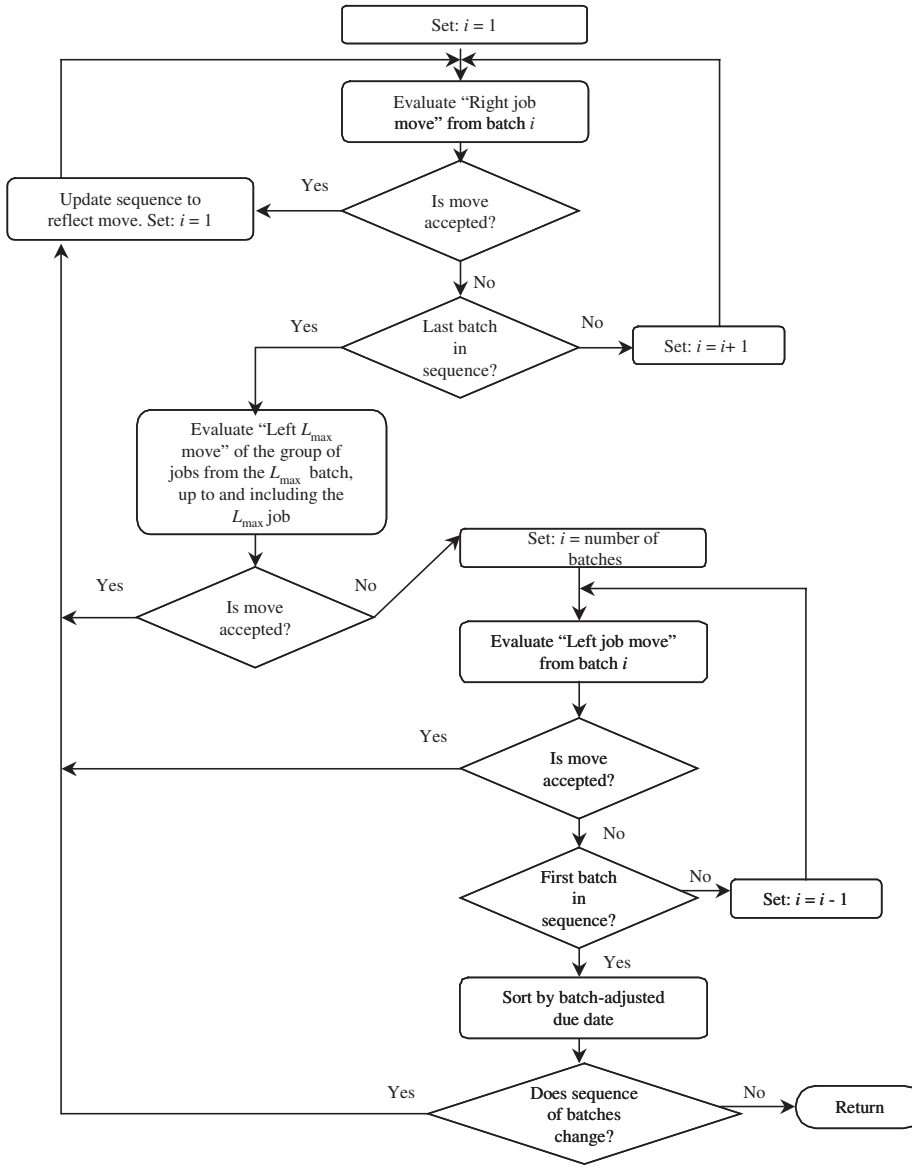


Figure 2. Procedure to find the 'best' schedule for a fixed batch partition.

Figure 2 describes the heuristic procedure. First, all right job moves are tested, starting with the first batch. Next, the left $L_{\max}$ move is evaluated. If accepted, the procedure starts over with right job moves. If not, left job moves are evaluated starting with the last batch. If any left job move is accepted, the whole procedure begins again. When all moves are tested, a final batch sort is made. If the batches are not re-sequenced, the procedure terminates. This heuristic requires $O(N-B)^3(B)^2$ time in the worst case, where $N = \sum_i N_i$ is the number of jobs and $B = \sum_i n_i$ is the number of batches. See appendix A for the discussion on the complexity of the fixed batch-partition heuristic.

Note that the $L_{\max}$ job in a sequence may not be in the subset containing the $l_{\max}$ and $l_{\max'}$ jobs. Thus, while the right and left moves appear to concentrate on subsets of jobs that do not directly affect the $L_{\max}$ of the sequence, eventually the subset containing the $L_{\max}$ job will be evaluated. Also, the focus on subsets not containing $L_{\max}$ allows for improved sequencing of those batches, which in turn allows for possible re-sequencing of the schedule as a whole that might not have occurred if only the subset containing the $L_{\max}$ job is evaluated.

Also note from figure 2 that while immediately accepting a right or left job move when $l_{\max}$ or $l_{\max'}$ is improved may result in a violation of condition 2 (earliest due-date by batch-adjusted due date), the condition, however, is met with a batch sort in the last action box in the flow diagram.

3. Batch-partition heuristic

In this section, a procedure for finding optimal/near-optimal solutions to the $1/s_{i,b}/L_{\max}$ problem is presented using insight gained from the optimality conditions.

3.1. *General algorithm methodology*

The methodology involves solving a series of constrained versions of the problem, i.e. fixed batch partitions. For the batch partition considered, a schedule is found that 'best' minimizes $L_{\max}$ using the fixed batch-partition heuristic described above. The algorithm proceeds from one batch partition to another using a constructive neighbourhood search that is described below. The search begins at partition $(1, 1, \ldots, 1)$ and continues until no neighbour is found that improves $L_{\max}$.

3.2. *Problem reduction*

Before constructing the sequence corresponding to the $(1, 1, \ldots, 1)$ partition, the problem is reduced using Theorem 3 from Hariri and Potts (1997). It identifies conditions where an optimal solution exists such that pairs of jobs j and k from the same family i are scheduled consecutively, i.e. j followed by k. When these conditions hold, the two jobs are replaced by a composite job with the processing time of the composite job being the sum of the two jobs' processing times. The due date of the composite job is set to $\min\{d_{ij} + p_{ik}, d_{ik}\}$.

The conditions under which a composite job is formed are as follows: for any family i, if $s_i + d_{i,1} \geq d_{i,2} - P_{i,2}$, where job $(i, 1)$ is the job with the earliest due date from family i, then there exists an optimal solution in which jobs $(i, 1)$ and $(i, 2)$ are scheduled contiguously; and if $d_{i,j} \geq d_{i,j+1} - P_{i,j+1}$ for any j $(j = 2, \ldots, N_{i-1})$, then there exists an optimal solution in which jobs (i, j) and $(i, j+1)$ are scheduled contiguously.

3.3. *Neighbour construction phase*

Neighbour construction consists of generating neighbouring sequences of an existing sequence. The objective of creating a neighbour is to reduce $L_{\max}$. The following three neighbour types are defined for the batch-partition heuristic.

- Neighbour type 1: consider a sequence S with the $L_{\max}$ job in batch G_{j1} (figure 3a). A neighbouring partition, S', is created by moving the last job from a batch of a differing family preceding G_{j1} to a position immediately after the $L_{\max}$ job (figure 3b). This increases the number of batches for families i and j by one.

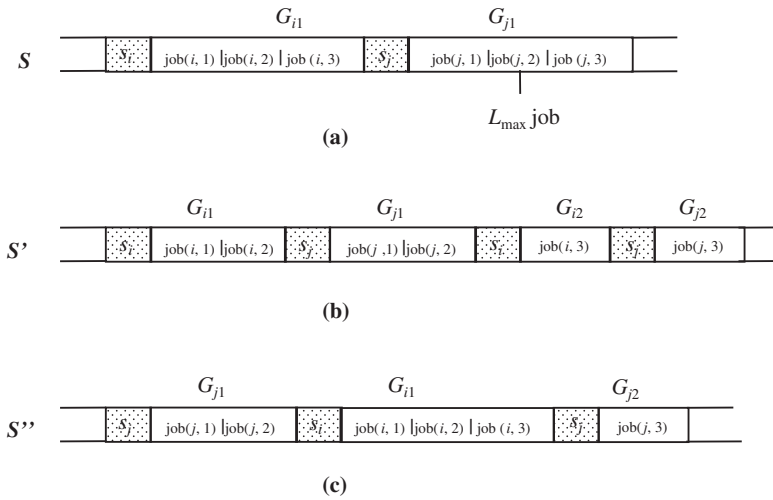


Figure 3. Generation of neighbours.

- Neighbour type 2: if the last job from batch G_{i1} was inserted after the last job in batch G_{j1} then the neighbouring partition would have an increase of one in the number of batches for only family i. Note that if the batch following G_{j1} is also of family i, then the number of batches does not increase. In this situation, the neighbour is not created because the fixed-batch-partition heuristic already accounts for this as a 'right job move'.
- Neighbour type 3: this neighbour has an additional batch for the part family containing the $L_{\max}$ job. It is generated from sequence S by forming a new batch consisting of the jobs from the $L_{\max}$ batch up to and including the $L_{\max}$ job. The new batch is placed in front of the batch preceding the $L_{\max}$ batch. Sequence S'' results when the jobs from G_{j1} in sequence S, up to and including the $L_{\max}$ job are inserted in front of batch G_{i1} (figure 3c). Note again, this neighbour is not created if the batch immediately preceding this newly created batch is from the same family. This condition is accounted for as a 'left $L_{\max}$ move' in the fixed-batch-partition heuristic.

Note that the generation of neighbour types 1–3 will lead to three neighbours of S if the $L_{\max}$ job is not the last job in a batch, but will lead to only two neighbours if the $L_{\max}$ job is the last job in a batch (neighbour types 1 and 2 are equivalent when the $L_{\max}$ job is the last job in a batch).

3.4. *Summary of the batch-partition heuristic*

The batch-partition heuristic begins by constructing a batch sequence in which all jobs from the same family are in a single batch, i.e. $(1, 1, \ldots, 1)$. The $(1, 1, \ldots, 1)$ partition is chosen as the initial solution because the neighbourhood structure is one of splitting batches, not combining batches, thus starting at a partition other than $(1, 1, \ldots, 1)$ may result in an inferior solution. Baker (1999) calls this $(1, 1, \ldots, 1)$ partition the GT sequence. All neighbouring sequences of the GT sequence are constructed and sequenced using the fixed batch-partition scheduling heuristic. Those sequences with $L_{\max}$ less than the $L_{\max}$ of the GT sequence are added to

the candidate list. Sequences are removed from the candidate list in FIFO order. Upon removal, if the sequence's $L_{\max}$ value is less than the incumbent value ($L^*_{\max}$), the neighbours of the sequence are generated. Otherwise, the sequence is discarded. As a neighbour is generated, the 'best' sequence is found using the fixed batch-partition scheduling heuristic. If $L_{\max}$ for the new sequence is less than the incumbent best $L_{\max}$ ($L^*_{\max}$), the neighbours of the new incumbent schedule are added to the candidate list and $L^*_{\max}$ is updated. Otherwise, the sequence is discarded. The process continues until the candidate list is empty. Figure 4 summarizes the batch-partition procedure.

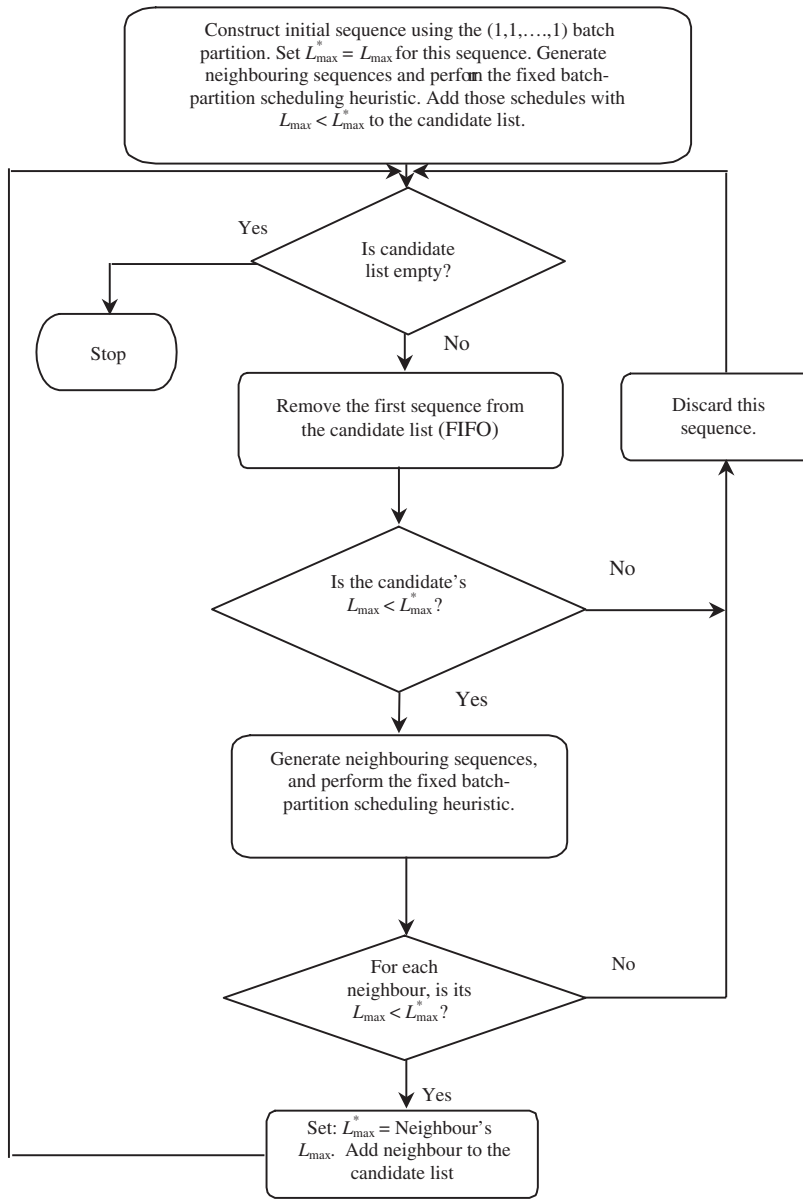


Figure 4. Batch-partition heuristic.

This batch-partition heuristic requires $O(F^2N^{5.6})$ time to solve, where F is the number of families and N is the number of jobs. See appendix B for a discussion on the complexity analysis.

4. Computational experiments

Three different experiments are conducted to evaluate the performance of the batch-partition heuristic. The first is modelled after experiments by Baker (1999) and is used to evaluate the differences in performance versus the Baker heuristic. The second is modelled after experiments by Hariri and Potts (1997) and is used to compare solution quality and computational effort against an optimal procedure. Finally, experimentation is performed to evaluate computational effort for large-scale problems.

4.1. *Experiment 1*

In the Baker framework, two problem sets are generated. One set consists of 24 or 25 jobs using problem size combinations (families × jobs/family) of 2×12, 3×8, 4×6 and 5×5. The other set consists of 36 jobs using 2×18, 3×12, 4×9 and 6×6 combinations. Processing times (p_{ij}) are generated randomly as Uniform [1, 99]. Due dates (d_{ij}) are generated as Uniform [0, $rn\bar{p}$], where r is a due date range, n is the number of jobs and $\bar{p}$ is the average processing time. Then, the smallest due date is subtracted from all due dates, making the minimum due date zero, and forcing a positive value for $L_{\max}$. All families have the same set-up time (ms^*). Baker defines m as a set-up time parameter and s^* as:

$$s^* = \max_i [\max_j \{d_{ij} + q_{ij}\} - \min_j \{d_{ij} + q_{ij}\}],$$

where q_{ij} is the sum of the processing times of all jobs from family i with due dates greater than that of job j.

Baker shows that the difficult region of problems to solve includes the following combinations of due date range (r) and set-up time (m) parameters indicated by XX:

		r			
		0.5	1.0	2.0	4.0
	0.10	XX	XX		
	0.25	XX	XX	XX	
m	0.50		XX	XX	XX
	0.75			XX	XX

Baker defines a performance measure for comparing the heuristic result to the optimal solution. Let the heuristic produce a solution $L_{\max}$, and let the optimal solution be $L^*_{\max}$, then:

$$V = (L_{\max} - L^*_{\max})/\bar{p}.$$

The performance measure V returns a value of 0 if the heuristic finds the optimal solution, and a value less than 1 if $L_{\max}$ is within an average processing time of the optimal solution.

Baker obtained his best results using the hybrid heuristic. This heuristic uses the best result of either his Gap/CS or GT/SC heuristic. The Gap/CS heuristic begins by evaluating jobs in earliest due date ordering and placing them one at a time into a sequence. Jobs are evaluated for a gap condition that suggests that jobs are either

placed at the end of the sequence or added to the latest batch of the same family. Once construction is complete, a neighbourhood search is initiated based on combining (C) and splitting (S) of batches and sorting batches by batch due date. The GT/SC heuristic starts with single batches for each family and sorts these batches by batch due date. The GT/SC also uses neighbourhood search, this time splitting (S) and combining (C) batches and sorting by batch due date.

Twenty problems were generated for each due date range and set-up time parameter combination for each problem, giving 800 test problems for each of the two problem sets. Optimal solutions were found using an implicit enumeration procedure. Table 1 shows that the batch-partition heuristic provides the optimal solution for at least 87% of the test problems in the first problem set. The average worst case performance is within $0.10^*\bar{p}$ (or 10% of an average processing time) of the optimal value. Table 2 shows that the heuristic provides the optimal solution at least 83% of the time for the second problem set. The average worst-case performance to objective is $0.18^*\bar{p}$. In both cases, the problems are constructed from the same parameter set. However, Baker's performance data are taken directly from his paper. CPU times are recorded indicating on average all problems solve under 0.05 s. The heuristic was coded in C++ and experiments were performed on a 500-MHz Pentium processor.

Tables 3 and 4 show how the heuristic performs over the due date and set-up time ranges. Note that the batch-partition heuristic consistently outperforms Baker's hybrid. It does very well when the set-up times are large, finding the optimal solution for all the problems when m is 0.5 and 0.75. An explanation for this observation is that with large set-up times, the optimal solution tends toward the group technology (GT) solution, and because the batch-partition heuristic begins its neighbourhood

Measure	Size	Baker's hybrid	Batch partition
Per cent optimal	2×12	66%	95%
	3×8	50%	88%
	4×6	48%	87%
	5×5	59%	89%
Average V	2×12	0.60	0.03
	3×8	0.90	0.10
	4×6	0.92	0.07
	5×5	0.52	0.08

Table 1. Summary of performance (Baker's first problem set).

Measure	Size	Baker's hybrid	Batch-partition
Per cent optimal	2×18	46%	92%
	3×12	35%	88%
	4×9	30%	86%
	6×6	41%	83%
Average V	2×18	1.20	0.11
	3×12	1.33	0.14
	4×9	1.03	0.16
	6×6	0.77	0.18

Table 2. Summary of performance (Baker's second problem set).

Set-up factor (m)	0.1	0.25	0.5	0.75
Average V – Baker's hybrid	0.36	0.71	0.76	0.31
Average V – Batch partition	0.12	0.16	0.00	0.00
Due date range (r)	0.5	1	2	4
Average V – Baker's hybrid	0.15	0.64	0.78	0.6
Average V – Batch partition	0.01	0.11	0.12	0.00

Table 3. Summary of performance (Baker's first problem set).

Set-up factor (m)	0.1	0.25	0.5	0.75
Average V – Baker's hybrid	0.64	1.32	1.58	0.42
Average V – Batch partition	0.28	0.31	0.00	0.00
Due date range (r)	0.5	1	2	4
Average V – Baker's hybrid	0.3	1.25	1.46	1.06
Average V – Batch partition	0.01	0.17	0.24	0.08

Table 4. Summary of performance (Baker's second problem set).

search at the GT solution, it tends to perform well for those problems with long set-up times. The heuristic also performs well when due dates are sampled from the smallest and largest range of the experimental design. Baker (1999) notes that when due dates are identical, it is desirable to have the minimum number of batches and that the GT solution achieves the minimum $L_{\max}$; thus with the batch-partition heuristic beginning its neighbourhood search at the GT solution, it performs well for problems where due dates are sampled from a small range. Also, as due dates become widely dispersed, the solution that minimizes $L_{\max}$ tends to be the due date ordering of all jobs. Therefore, problems with large due date ranges tend to be well solved by the batch-partition heuristic because the heuristic sorts jobs within a family by earliest due date, and continually divides batches until no improvement in $L_{\max}$ is observed, finally rendering due date ordering of all jobs.

4.2. *Experiment 2*

In this experiment, the batch-partition heuristic is assessed in relation to the Hariri and Potts (1997) branch-and-bound algorithm. Hariri and Potts generate test problems with 30, 40, 50 and 60 jobs (N) and with 2, 4, 6, 8 and 10 families (F). Jobs are distributed uniformly across families, with the last family containing additional jobs when the number of families does not divide evenly into the number of jobs. Processing times are random integers from the Uniform distribution [1, 100]. Having generated processing times, the sum of the processing times is $P = \sum_{i,j} p_{ij}$. Due dates are generated as random integers from the Uniform [a*P, b*P], where a takes on the values {0.0, 0.2, 0.4, 0.6, 0.8} and b takes on the values {0.2, 0.4, 0.6, 0.8, 1.0}, with a < b. This results in 15 combinations of due date range experiments. Set-up times are integers generated from the following Uniform distributions:

- Class A: [1, 100].
- Class B: [1, 20].
- Class C: [101, 200].

		Average computation time (s)			Average number nodes			Number of unsolved problems			Number of problems for which LB = UB		
		Set-up class			Set-up class			Set-up class			Set-up class		
N	F	A	B	C	A	B	C	A	B	C	A	B	C
30	2	0.01	0.01	0.02	13	6	22	0	0	0	45	53	45
	4	0.10	0.13	0.18	116	146	196	0	0	0	19	24	12
	6	0.17	0.19	0.32	188	206	342	0	0	0	9	16	2
	8	0.11	0.10	0.26	113	101	277	0	0	0	11	15	3
	10	0.10	0.09	0.21	96	90	209	0	0	0	8	9	3
40	2	0.06	0.04	0.06	61	41	57	0	0	0	39	44	38
	4	0.50	0.71	1.06	554	754	1100	0	0	0	21	21	10
	6	0.79	1.33	3.28	856	1407	3370	0	0	0	9	16	0
	8	0.71	0.81	3.67	740	839	3635	0	0	1	2	7	1
	10	0.74	0.80	2.23	769	811	2232	0	0	0	5	9	2
50	2	0.06	0.11	0.12	63	116	132	0	0	0	43	43	41
	4	2.54	3.07	5.70	2741	3219	5952	0	0	2	19	21	14
	6	6.28	10.06	11.03	6607	10298	11155	1	3	3	8	10	0
	8	4.02	7.36	17.66	4150	7394	17786	1	1	9	2	4	2
	10	4.87	9.79	19.00	4885	9655	18716	1	1	7	1	2	0
60	2	0.96	0.27	0.70	1092	297	751	0	0	0	40	45	37
	4	5.78	11.77	10.35	6163	12051	10375	2	5	5	18	18	14
	6	11.82	21.38	25.00	12150	21684	26022	3	8	12	4	4	2
	8	18.60	30.99	31.47	18608	30714	43655	9	16	17	3	3	1
	10	17.46	27.26	34.38	17047	26547	34792	7	10	20	1	1	0

Table 5. Performance of Hariri and Potts' algorithm.

Note that class A has set-up times in the same range as the processing times, class B has relatively small set-up times and class C has set-up times larger than any processing time. Five test problems are generated for each due date range, for 75 test problems for each combination of job size, family size and set-up class, for a total of 4500 problems. Table 5 is a copy of results presented by Hariri and Potts. They were able to solve all but 144 problems in 100 s on an IBM 3090. Their results also present the average solution time and average number of nodes evaluated. If a problem was abandoned due to computation time, they used the time and number of nodes at the time of abandonment for the average time and node calculations. In the last column, they present the number of times problems were solved at the root node. They summarize that their branch-and-bound algorithm is effective in solving problems with up to 50 jobs.

Table 6 is an evaluation of the performance of the batch-partition heuristic over this same experimental framework. The batch-partition heuristic is coded in C++ and executed on a 500-MHz Pentium processor. Optimal solutions for each problem were obtained using an implicit enumeration procedure. Problems are classified as unsolved if the implicit enumeration was unable to obtain an optimal solution. Average computation times and average number of nodes visited are presented for each job, family and set-up class combination. The average number of nodes visited represents the average number of neighbouring fixed batch-partition solutions evaluated. Also displayed is the number of problems in which the batch-partition

		Average computation time			Average number nodes			Number of unsolved problems			Number of problems solved optimally		
		Set-up class			Set-up class			Set-up class			Set-up class		
N	F	A	B	C	A	B	C	A	B	C	A	B	C
30	2	0.01	0.01	0.01	7	8	5	0	0	0	71	70	74
	4	0.02	0.02	0.01	24	32	15	0	0	0	62	49	70
	6	0.03	0.04	0.02	49	61	23	0	0	0	56	48	69
	8	0.03	0.05	0.02	50	82	24	0	0	0	65	59	74
	10	0.07	0.08	0.06	83	129	32	0	6	0	59	41	69
40	2	0.01	0.01	0.01	7	9	5	0	0	0	71	68	74
	4	0.03	0.03	0.02	28	38	19	0	0	0	53	52	60
	6	0.05	0.08	0.03	61	86	33	0	2	0	58	43	66
	8	0.10	0.11	0.07	100	140	51	1	5	0	45	36	64
	10	0.08	0.14	0.08	117	178	56	11	23	0	34	17	61
50	2	0.02	0.02	0.02	7	9	5	0	0	0	69	66	73
	4	0.04	0.04	0.02	34	46	21	0	0	0	49	48	63
	6	0.08	0.11	0.05	72	107	44	0	3	0	40	35	59
	8	0.15	0.21	0.11	118	187	62	4	15	0	29	0	48
	10	0.18	0.28	0.11	170	254	89	18	24	3	23	10	52
60	2	0.03	0.03	0.03	8	10	6	0	0	0	67	71	70
	4	0.06	0.06	0.06	36	49	23	0	0	0	46	44	55
	6	0.12	0.13	0.06	83	118	50	1	3	0	37	30	49
	8	0.20	0.27	0.13	146	206	80	10	19	2	29	24	47
	10	0.31	0.41	0.20	222	308	120	31	40	8	14	7	34

Table 6. Performance of a batch-partition heuristic.

heuristic obtained the optimal solution. Of the 4500 problems, the heuristic obtained 3053 optimal solutions, 1218 solutions were suboptimal, and it was indeterminate for the remaining 229 problems because the implicit enumeration was abandoned.

While the heuristic obtains a significant number of optimal solutions, the branch-and-bound algorithm can obtain optimal solutions for nearly all problems in the experimental framework. The trade-off, however, is that the branch-and-bound solution time is increasing at a substantially greater exponential rate and, as the authors point out, is effective in only solving problems with up to 50 jobs. Of additional interest is that while the heuristic obtained optimal solutions for approximately two-thirds of the problems, the average V for the heuristic over all problems (except those where the enumeration was abandoned) was 0.14. In other words, the batch-partition heuristic found optimal or near-optimal solutions for all problems.

One may conclude from this experiment that an approach to solving these problems is to begin with the Hariri and Potts branch-and-bound algorithm, and if unable to solve after a time, resorting to the batch-partition heuristic which should return a near-optimal solution in a relatively short time.

Also of interest is that Hariri and Potts' branch-and-bound algorithm found the Class C set-up problems to be the most challenging while the batch-partition heuristic performed best on these problems. Performance was best for Class C problems because again, as observed in Experiment 1, the batch-partition heuristic performs best on those problems with large family set-up times.

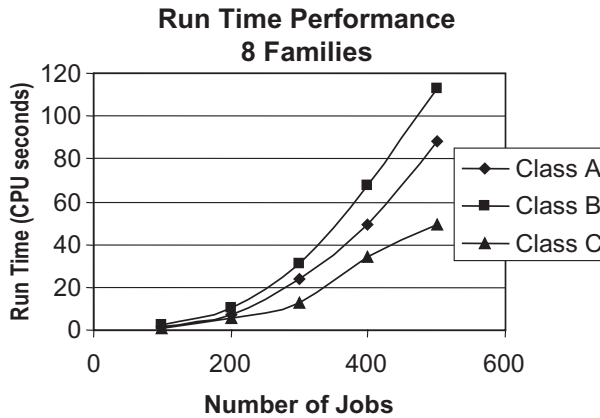


Figure 5. Batch-partition heuristic solution times by set-up class for problems with eight families.

4.3. *Experiment 3*

Since the computational cost of the batch-partition heuristic is non-polynomial, a third series of experiments are performed to evaluate the size of the problem the heuristic is capable of solving. Tests are run for the problems using the three set-up classes and for two, four, six and eight families. Again, 75 test problems are generated over the due date ranges for each job size, family size and set-up class combination using the Hariri and Potts experimental framework. The job sizes range from 100 to 500 jobs in increments of 100. Figure 5 represents the average solution times for the eight family problem sets and the three set-up classes (depicted as A, B and C on the legend). Figure 5 indicates that the batch-partition heuristic solves problems for up to 500 jobs in less than 2 min. For the two-, four- and six-family problems, the batch-partition heuristic solves problems for up to 500 jobs in less than 1, 12 and 50 s, respectively. When scaled, the performance graphs for the two-, four- and six-family problems are similar to figure 5.

5. Conclusions

The $1/s_{i,b}/L_{\max}$ problem is NP-hard. The best algorithms in the literature for finding optimal solutions have been successful in solving problems up to a size of 60 jobs. An effective heuristic that produces optimal/near-optimal solutions for the problem was presented. The batch-partition heuristic was shown to find optimal solutions for 88 and 85% of problem instances in two experiments. In addition, over a range of problem instances, the heuristic is shown to find solutions for problems with up to 500 jobs and eight part families in less than 2 min.

Acknowledgements

Research was supported, in part, by the Furniture Manufacturing & Management Center, North Carolina State University, and by the Office of Naval Research, Contract #N00014-90-J-1009.

Appendix A: Complexity of the fixed batch-partition heuristic

Considering the heuristic shown figure 2, there are three primary operations: 'right job move', the 'left $L_{\max}$ move' and the 'left job move'. Whenever a job

move is made, the algorithm restarts. Therefore, the overall order is the product of the order of each operation. Note that a fourth operation, batch sort, in practice rarely results in a new sequence of jobs and therefore has essentially no impact on the problem complexity.

- Order of right job move: because the number of batches is fixed, at least one job must remain in each batch. Therefore, in the worst case, $N-B$ jobs can be moved, where N is the number of jobs and B is the number of batches. Therefore, the order of the right job move is $O(N-B)$.
- Order of left $L_{\max}$ move: a left $L_{\max}$ move can be executed, or at least assessed, for all batches except for the first two batches if the batch-partition integrity is to be maintained. In addition, a batch sort, of the order $O(B)$, is performed for each move. Therefore, the order of the left $L_{\max}$ move is $O(B)(B-2)$.
- Order of left job move: similar to the argument for the right job move, in the worst case, $N-B$ left job moves can be executed. However, to evaluate the acceptance of this move, each batch within the subgroup must be tested. In the worst case, $N-B-2$ of these batches will be evaluated. Therefore, the order of the left job move is $O(N-B)(N-B-2)$.

The order for the fixed batch-partition heuristic is the product of the three primary operations: $O(N-B)^2(B-2)(N-B-2)B$, or simply $O(N-B)^3(B)^2$.

Appendix B: Complexity of the batch-partition heuristic

From appendix A, the order of the fixed-batch partition is $O(N-B)^3(B)^2$. However, the batch-partition heuristic searches over a series of fixed batch partitions of varying batch sizes (B). The worse-case order for the fixed batch-partition heuristic occurs when $B = 2N/5$, observed by taking the first derivative of $(N-B)^3(B)^2$ with respect to B and setting this equal to 0. Therefore, the worse-case order for the fixed-batch partitions visited is $O(N-(2N/5))^3(2N/5)^2$, or simply of order $O(N^5)$.

The batch-partition heuristic performs a series of these fixed batch-partition heuristics. Table 7 presents the observed maximum number of fixed batch partitions visited for the 75 problem sets generated for various job and family-sized problems

		Maximum				Maximum	
N	F	Number of neighbours	$F^2N^{0.6}$	N	F	Number of neighbours	$F^2N^{0.6}$
50	2	29	42	300	2	60	123
50	4	179	167	300	4	542	490
50	6	329	376	300	6	1113	1103
50	8	591	669	300	8	1961	1961
50	10	1056	1046	300	10	2919	3064
200	2	55	96	400	2	72	146
200	4	415	384	400	4	671	583
200	6	904	865	400	6	1210	1311
200	8	1615	1537	400	8	2241	2330
200	10	3161	2402	400	10	3516	3641

Table 7. Observed maximum number of neighbouring batch partitions visited.

using Hariri and Potts' experimental framework described in Experiment 2. The observed maximum number of fixed batch partitions visited is approximately $F^2N^{0.6}$, where F is the number of families and N is the number of jobs.

Because the batch-partition heuristic performs a series of fixed batch-partition heuristics, the complexity of the batch-partition heuristic is a function of the number of fixed batch partitions and the complexity of the fixed-batch-partition heuristic: $O(N^5)(F^2N^{0.6})$, or simply $O(F^2N^{5.6})$, where N is the number of jobs and F is the number of families.

References

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

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

Bruno, J. and Downey, P., 1978, Complexity of tasks sequencing with deadlines, set-up times and changeover costs. *SIAM Journal of Computing*, **7**, 393–404.

Hariri, A. M. A. and Potts, C. N., 1997, Single machine scheduling with batch set-up times to minimize maximum lateness. *Annals of Operations Research*, **70**, 75–92.

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

Potts, C. N. and VanWassenhove, L. N., 1992, Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of the Operational Research Society*, **43**, 395–406.

Schutten, J. M., Van de Velde, S. L. and Zijm, W. H., 1996, Single machine scheduling with release dates, due dates and family setup times. *Management Science*, **42**, 1165–1174.

Unal, A. T. and Kirnan, A. S., 1992, Batch sequencing. *IIE Transactions*, **24**, 73–83.

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