PERGAMON

Omega, Int. J. Mgmt. Sci. 27 (1999) 485–495

omega
The International Journal of Management Science

An improved heuristic for the single-machine, weighted-tardiness problem

J.E. Holsenback [a,*], R.M. Russell [b], R.E. Markland [c], P.R. Philipoom [c]

[a] College of Business Administration, Savannah State University, Savannah, GA 31404, USA
[b] Faculty of Management, The University of Calgary, 2500 University Drive, N.W., Calgary, Alberta., Canada T2N 1N4
[c] College of Business Administration, The University of South Carolina, Columbia, SC 29208, USA

Received 1 August 1996; accepted 1 November 1998

Abstract

This research considers the single machine, total weighted-tardiness problem. We propose a method for modifying due dates that eases the construction of optimal schedules. We employ due date modification in a new heuristic that is fast, solving 50-job problems on an IBM 4381 in at most 0.14 CPU s. The heuristic is shown to be superior to previously known heuristics in minimizing total weighted-tardiness, especially in settings where 40% or more of the jobs can be completed without tardiness on the single machine.

Keywords: Scheduling; Heuristic; Weighted tardiness; Single machine

1. Introduction

The problem of completing a job as close as possible to a promised delivery date is of primary importance to operations managers. As a result, research into scheduling jobs for processing, under a wide variety of shop conditions, has long occupied a prominent place in production and operations management literature. By far, the largest portion of this research has focused on the development and testing of dispatching rules. A dispatching rule is a method of selecting the next job for processing from the unscheduled list of jobs available. If one considers a queue of jobs to represent an unscheduled list, then repeated application of a dispatching-rule results in a *schedule*, wherein the starting and completion time of each job has been determined.

While few plants are comprised of a single machine that continuously processes orders, many plants behave like a single machine. In linear flow operations, such as an assembly shop, where operations are sequentially performed on a batch of product, a capacity constrained operation, or bottleneck, is frequently encountered. Scheduling of the bottleneck, then, effectively schedules the entire operation.

Typically, each operating department has a group of jobs available for processing at any one time and it is the task of a scheduler to specify the order of processing for these jobs, consistent with some predetermined measure of performance. As some jobs are completed and new ones arrive at the work center, schedules must be revised. The scheduler may antici-

* Corresponding author. Tel.: +1-912-353-3142; fax: +1-912-356-2837; e-mail: edward.holsenback@savstate.edu

0305-0483/99/$ - see front matter
PII: S0305-0483(98)00072-3

pate these dynamic arrivals, but runs a risk of infeasibility when delays occur and a scheduled job is not available to begin processing at its assigned time. Rather than anticipate job arrivals, it would seem preferable to revise the schedule as each job actually arrives at the work center. While an operation (machine) can process only one job at a time, it may be beneficial to consider the relative sequencing of *all* jobs in selecting the next job for processing. The consideration of all the jobs will increase the number of available options to be evaluated, with which judicious trade-off analysis will hopefully lead to better, and less-myopic, solutions. Despite the potential for better solutions, the scheduling method must be fast in order to revise the incumbent schedule, periodically, as jobs are completed and new jobs have arrived.

The (repeated) application of static scheduling methods in the inherently dynamic nature of the real world raises the question whether the consideration of all the scheduling information at a particular point in time is beneficial. A study by Blackburn and Millen [2] showed that an optimal routine might not perform better than a heuristic when implemented, periodically, in an environment with a rolling horizon. As a practical matter, the development of scheduling algorithms must be static in nature if there is to be any rigorous comparison of their relative performance. Moreover, the focus of this current research is on the development and testing of new heuristic methods. Evaluating static algorithms, both heuristic and optimal, in a dynamic environment could be a topic for future research.

The effectiveness of a schedule generated by using a dispatching rule or scheduling procedure can be measured in a variety of ways. In dynamic environments, two of the more commonly used measures are percent of tardy jobs and average job tardiness. The evaluation of tardiness is straightforward; a job is tardy if it is completed after its assigned completion time. It is also possible to assign a penalty cost to each unit of a job's tardiness. These penalties, referred to as job weights, reflect losses to the shop for failure to complete a job on time. In the case of unweighted tardiness, we assume that all jobs have equal importance; otherwise, job weights can be used to differentiate the relative importance of the job or customer. Tardiness may be penalized linearly, such that each unit of tardiness for a particular job incurs the same penalty, or through a quadratic function, where greater tardiness is assessed increasingly larger weights. In either case, the penalties must be non-decreasing, as they would, otherwise, lead to an unrealistic situation where it is beneficial to allow a job to become increasingly tardy.

With a large queue of jobs, it becomes extremely difficult for the scheduler to ensure that a particular schedule is 'best'. When there are n jobs to be scheduled on a single machine and there is no inserted idle time, nor job preemption, 2^n sequences or schedules must be evaluated to determine an optimal schedule. In a study by Russell and Holsenback [14], they report that as machine utilization approached 90 percent, queue lengths in an open job shop often exceeded 35 jobs. This combinatorial problem has been shown by Lawler [7] to be NP-complete.

1.1. Problem description and assumptions

For the problem of sequentially ordering N jobs, processed on a single machine, we make the following standard assumptions. A job, J_i, has a known integer processing time, p_i (including set-up and knockdown times) and an integer due date, d_i, measured from time zero, which is the time when all jobs are available for processing. Additionally, each job has an integer weight, w_i, that reflects a linear penalty assessed for each time unit that the job is completed after its due date. The machine has no prior commitments and is available for continuous processing. Finally, the sequencing of any job is independent of other jobs (i.e. set-up times are not sequence dependent) and jobs are not preempted.

Throughout our discussion we assume that a set of N jobs, each with a corresponding processing time and nominal due date, is mapped into some processing sequence $(J_1, J_2, \ldots, J_N)$. At each iteration of the algorithm, the position of jobs within the sequence may change. For convenience, the processing time and due date of a job correspond with the position of the job in the current sequence, thus p_4 and d_4 are the processing time and due date of the fourth job in the incumbent sequence. Failing to complete the job on or before its due date results in a penalty that is weighted by w_i and applies equally to each unit of the job's tardiness. Typically, one could assume that the job due date is measured in hours or days, and the job weight represents the penalty, in monetary units, per due date measure.

The problem is to find a schedule for N jobs such that we minimize total weighted tardiness, where the tardiness, T_i, of a job, J_i, is weighted by some factor w_i and summed over the N jobs under consideration. Thus, $WT = \Sigma_{i=1} w_i T_i$, where T_i is the positive difference between the job's completion time, C_i, and its nominal due date, where $C_i = \Sigma_{j=1} p_j$ and $T_i = \max(0, C_i - d_i)$. When J_i is not tardy, job slack, S_i, is given by $S_i = \max(0, d_i - C_i)$.

2. Previous research

McNaughton [8] used the ratio r_i, where $r_i = w_i/p_i$, to develop rules regarding the relative positioning of jobs in an optimal sequence. He showed that if a schedule exists in which two adjacent jobs, J_j and J_k (j preceding k) are both tardy and cannot be completed early regardless of their respective ordering, the job with the lowest r_i must be scheduled last. In other words, if $C_j > d_j$, $C_k - p_j > d_k$ and $r_k < r_j$, then J_j must precede J_k in an optimal schedule. A schedule constructed by sequencing jobs in order of non-increasing r_i is referred to as a weighted shortest processing time (WSPT) schedule. McNaughton further showed that, even without consideration of set-up time, there is no advantage to be gained in job splitting. Therefore, the construction of an optimal schedule depends solely on the processing sequence.

Elmaghraby [4] formulated a network model that is equivalent to backward recursion in a dynamic programming algorithm. He sequentially built a schedule from back to front, first determining the last job in a schedule, then the next-to-last job, and so on until the first job had been scheduled. He used three lemmas to confine the solution space. Of these, lemma 1, generally referred to as Elmaghraby's lemma, is usually incorporated into algorithms that form schedules from the back to the front (e.g. Refs. [12, 15, 16]). This lemma states that a job is scheduled last in an optimal sequence if it can be completed without tardiness.

Rinnooy Kan et al. [13] formalized four theorems for the general non-decreasing cost function and stated these as corollaries for the weighted cost function. Their Corollary 1 provides a stronger form of Shwimer's [17] precedence constraint and yields a dominance property when we can identify a set of jobs, B_k (the 'before k' set), that can be shown to precede J_k in at least one optimal schedule. This corollary states that given $w_j > w_k$ and $p_j < p_k$ and $d_j \leq \max(d_k, (\sum_{i \in B_k} p_i + p_k))$, we need only consider schedules wherein J_j precedes J_k.

Baker and Schrage [1] showed task chains to be a particularly efficient method of incorporating dominance properties in a dynamic programming algorithm. A chain is a set of tasks in which each task has at most one direct predecessor and at most one direct successor. While dominance properties do not necessarily generate chain structures, subsets of jobs that form chains can be generated thereby reducing the storage space and computational effort required. The authors reported that the dynamic programming algorithm solved even the most difficult 20-job problems of Rinnooy Kan et al. [13] in an average time of 2.52 CPU s.

Schrage and Baker [16] devised a labeling procedure that can uniquely address each of the sequences that are generated while avoiding the necessity for 'decoding'. This labeling procedure assigns a unique code to a sequence, identifying its location within an array, enabling the sequence to be directly retrieved from storage. This procedure was shown to be superior to the chain algorithm of Baker and Schrage [1] in terms of solution time for both weighted and unweighted tardiness problems.

Morton and Rachamudugu [9] identified a new property for optimally sequencing adjacent jobs. This property, which they called Proposition 1, states that for any two adjacent jobs, J_j and J_k, J_j must precede J_k in an optimal sequence if the following holds:

$$\frac{w_j}{p_j}\left[1 - \frac{(d_j - t - p_j)^+}{p_k}\right]^+ \geq \frac{w_k}{p_k}\left[1 - \frac{(d_k - t - p_k)^+}{p_j}\right]^+ \quad (1)$$

In this expression, t represents the start time for J_j, given by C_{j-1} and $[f(z)]^+$ signifies $\max[0, f(z)]$. Unlike McNaughton's rule, Proposition 1 does not require both jobs to be tardy in either position and will optimally sequence adjacent jobs even if they can be completed without tardiness. Proposition 1 was used to develop a heuristic, identified by these authors as HR3, which, for the purpose of developing a schedule, assigns an apparent priority (AP) to each job, where $\mathrm{AP}_i = w_i/p_i \exp((-k/\bar{p})(d_i - t - p_i)^+)$. The schedule is created from front to back, with t defined as the earliest start-time for J_i. For the first job scheduled, t is equal to zero. As jobs are scheduled, both t and $\bar{p}$, the average processing of all remaining unscheduled jobs, are updated. The parameter k is set to a predetermined value between 0.5 and 2.0. Morton and Rachamudugu used $k = 0.5$ in their testing of HR3.

Potts and VanWassenhove [12] introduced a branch and bound algorithm using Lagrangian relaxation and a multiplier adjustment method to compute and update a lower bound. The method uses a heuristic to form an initial sequence and then chooses multipliers so that the heuristic solution also solves the Lagrangian problem. Shortest weighted processing time was selected to form the initial sequence and provide an upper bound on weighted-tardiness. Two well-known pruning devices were used to reduce the number of branches in the search tree. Adjacent pair interchange (API) was used since it is easily implemented and can be performed quickly.

Chambers et al. [3] proposed a heuristic which uses dominance properties and problem decomposition to quickly solve problems with up to 50 jobs. Two heuristic dominance rules were developed and shown to produce optimal sequences under certain conditions. These rules were then incorporated into a decomposition heuristic, referred to as DH(m, f), using dynamic programming and the Schrage–Baker [16] labeling technique. The heuristic removes m jobs from the

unscheduled set and solves this subset optimally, using the Schrage–Baker algorithm. The first f jobs in the solved subset are added to the scheduled set and replaced by a like number from the unscheduled set. The procedure continues until no jobs remain in the unscheduled set. The authors initially tested the heuristics of Schild and Fredman (SF) [15], Wilkerson and Irwin (WI) [18] and Morton and Rachamudugu (MR) [9] to determine which heuristic to use for further comparison.

Chambers et al. reported that the MR heuristic with $k = 0.8$ provided better results than either SF or WI. DH(15, 5) was then shown to be superior to MR with respect to percent of cost over optimal and the number of optimal solutions. For problems where an optimal solution was not known, the heuristics were simply compared with each other. The new heuristic was seen as a significant improvement over that of MR in terms of solution quality.

Progress in expanding the size of problems that can be solved optimally has come incrementally as new dominance properties have been identified and with improvements in computing hardware. While the well-known total tardiness problem can be quickly solved for problems with 100 jobs, current solution techniques for the weighted tardiness problem can struggle with problems of as few as 30 jobs.

3. Modified due dates

It is fairly common in industrial environments for a job to face the prospect of being tardy regardless of when it is processed. Such unavoidable tardiness will occur when a job is already tardy or when its processing time is greater than the remaining time until its due date. Starting at time zero, when $d_j \leq p_j$, the construction of an EDD sequence can easily result in demonstrably inferior sequences. Consider the following EDD sequence:

Job	p_i	d_i	w_i	T_i	w_iT_i
1	16	4	6	12	72
2	10	16	8	10	80
					152

This *sequence, however, violates the precedence constraint of Rinnooy Kan et al. [13] Since* $p_2 < p_1$, $w_2 > w_1$ and the sum of the processing times for the jobs that must precede J_1, $p(B_1) = 0$, then, for sequencing purposes, $d_1 = \max[d_1, (p(B_1) + p_1)] = \max[d_1, p_1] = \max[4, 16] = 16$. Setting d_1 to 16, for sequencing purposes, now requires that J_2 precede J_1 in an optimal sequence. Thus, when J_2 and J_1 are interchanged, we show a reduction in weighted tardiness, as seen below.

Job	p_i	d_i	w_i	T_i	w_iT_i
2	10	16	8	0	0
1	16	4	6	22	132
					132

Our Proposition A, described in below, provides a method of modifying due dates which avoids the problem, shown above, of starting with an inferior earliest due date sequence.

Proposition A. An optimal sequence is not altered by considering the modified due date, $\mathrm{dmod_j}$, of a job as the maximum of the job's processing time, $\mathrm{p_j}$, or its nominal due date, $\mathrm{d_j}$. Thus, $\mathrm{dmod_j} = \max(\mathrm{d_j}, \mathrm{p_j})$, where $\mathrm{dmod_j}$, represents a job's due date for sequencing purposes.

The proof of Proposition A, provided in Appendix A, is straightforward and relatively simple. All subsequent discussion regarding an EDD sequence will assume that $\mathrm{d_j} > \mathrm{p_j}$ for all jobs in the set to be scheduled.

4. A new heuristic

In this section, a new heuristic will be developed by sequentially selecting a job to be scheduled last within the set of unscheduled jobs. This selected job is then added to the scheduled set in the first position. As described earlier, this represents a method whereby the schedule is built from back to front. The last position in the unscheduled set, Φ_U, is referred to as position k. It is a relatively simple matter to show that if the scheduled set, Φ_S, represents an optimal sequence, then whichever job in the unscheduled set properly occupies k must also occupy the first position in the scheduled set. In this section, a method for selecting one of these jobs, J_j, will be described. This method calculates the net cumulative reduction in total weighted tardiness derived by removing J_j from Φ_U and adding it to Φ_S. Three heuristic rules will be used to avoid demonstrably inferior sequences in Φ_S. The heuristic utilizes an adjacent pair interchange technique to compute the weighted tardiness gain (or loss) in Φ_U derived from successively interchanging J_j and the following job, J_i, until $i = k$. All discussion of this new heuristic assumes that the unscheduled set is ordered EDD.

4.1. Pairwise interchanges

The cost, CI_{ji}, of interchanging two jobs, J_j and J_i, is the increase in tardiness of J_j multiplied by w_j. This cost is mitigated by the slack of J_j, S_j, before interchange and is given by:

$$CI_{ji} = \max(0, p_i - S_j)w_j \quad (2)$$

The gain accompanying interchange of the jobs is the reduction of weighted tardiness of J_j and is given by:

$$G_{ji} = \min(0, T_i - p_j)w_i \quad (3)$$

The net gain, which can be negative, derived from interchanging the two jobs, is:

$$NG_{ji} = G_{ji} - CI_{ji} \quad (4)$$

4.2. Successive pairwise interchanges

We can also iteratively compute the net cumulative gain, NCG_{jl}, of sequentially interchanging J_j with each succeeding job, J_i, up to and including some job, J_l. Thus, NCG_{jl} is the total change in weighted-tardiness derived from moving J_j from its position in the current sequence and placing it immediately after J_l. NCG_{jl} is given by:

$$NCG_{jl} = \sum_{i=j+1}^{l} NG_i \quad (5)$$

The overall gain, OG_j, from repositioning J_j to position k is simply NCG_{jk}. Thus, in evaluating whether it is beneficial for J_j to occupy the last position, k, in the unscheduled sequence, we find it useful to retain, for reference, the maximum value attained by NCG_{jl} for $l = j+1, \ldots, k$. If it is found that the maximum NCG_{jl} is greater than OG_j, then the total weighted tardiness of the sequence $\{J_{j+1}, J_{j+2}, \ldots, J_k, J_j\}$ is greater than some other sequence with J_k in the last position. Example 2 demonstrates the computation of each term along with the case where $\max(NCG_{jl}) > OG_j$. Note that $\max(NCG_{jl})$ can never be less than OG_j, since OG_j is an element of the set NCG_{jk}.

We now present the three rules to improve our heuristic performance in which, at each iteration, a job, J_j, is removed from Φ_U and placed in the first position in Φ_S.

Rule 1. Remove that job, J_j, from Φ_U and include it in the first position in Φ_S, subject to the provisions:

1. $OG_j = \max(NCG_{jl})$.
2. $OG_j > 0$.

If more than one job satisfies this property, then select for inclusion in Φ_S that job, J_j, having the greatest OG_j. If no job satisfies these conditions, proceed to Rule 2.

Rule 2. Select from Φ_U that job, J_l, having:

(a) $w_l < w_k$.
(b) $w_l/p_l = \min(w_i/p_i)$ for all jobs J_i, contained in Φ_U; subject to
(c) $p_l < T_k$.

(Note that requirement (c) is necessary since J_l and J_k cannot exist as an adjacent pair in an optimal sequence. It is *possible*, however, that J_l can follow J_k if some other job, J_m, lies between the two jobs. This situation will occur only when J_k remains tardy after repositioning J_l). Compute OG_l, remove J_l from the unscheduled set and for each J_j identified using Rule 1, recompute OG_j, subject to Rule 1. If $OG_j - OG_l > 0$ for any J_j, then include J_l in Φ_S.

Rule 2 enables us to consider that removing a job from the unscheduled set may create valuable slack in Φ_U in other jobs eligible to follow J_k in Φ_S. Accepting an increase in weighted tardiness due to the repositioning of J_l may ultimately lead to an overall reduction in total weighted tardiness.

Using Rules 1 and 2 can easily lead to demonstrably sub-optimal conditions in Φ_S. Consider that it is possible to add a job, J_j, to Φ_S such that one or more of the jobs in Φ_S. should precede J_j. Invoking Rule 3 at each iteration avoids this problem.

Rule 3. In adding the job, J_j, to Φ_S, where J_l is currently the first job in Φ_S, if the condition $[\min(T_l, p_j)]w_l > \max[0, (T_l + d_j - d_l)]w_j$ holds, we will interchange J_l and J_j. Now letting $l = l + 1$, we repeat the test until either:

(a) $[\min(T_l, p_j)]w_l < \max[0, (T_l + d_j - d_l)]w_j$ or
(b) $l = N$.

The term $\max[0, (T_l + d_j - d_l)]$ represents the resulting tardiness of J_j after interchange with J_l and the condition will apply only if $T_l > 0$. Rule 3 allows a new job to be added to Φ_S such that no adjacent pair violates the Morton and Rachamudugu rule.

Section 5 describes the new heuristic, referred to as the HMR algorithm.

5. The HMR heuristic algorithm

Step 1: modify job due dates as necessary, subject to Proposition A.

Step 2: order the set by non-decreasing due dates, where due date refers to the modified due date. If ties exist, order by non-decreasing processing times and if ties continue to exist, order by non-increasing weights (i.e. create an EDD sequence).

Step 3: compute slack and tardiness for each job in the current sequence.

Step 4: beginning with the last job in the current sequence, J_N, if $T_N = 0$, add J_N to Φ_S and continue until $T_i > 0$ for $i = N-1, \ldots, 1$. Note that if $T_i = 0$ for all i the solution is optimal.

Step 5: identify the set of jobs, J_j, eligible for addition to Φ_S using Corollary 1 of Rinnooy Kan et al. Beginning with J_1 in the current sequence, let $l = 1$ and $m = l + 1$. If $p_m > p_l$ and $w_m < w_l$, then HOLD(l) = m. Otherwise, $m = m + 1$, HOLD(l) = 0. Continue in this fashion until $m = k$ or HOLD(l) = m and $l = k-1$. Those jobs, J_j, with HOLD(i) = 0, along with J_k are eligible for addition to Φ_S.

Step 6: compute OG_j for the jobs identified in step 5. If no job meets the requirements of Rule 1, apply Rule 2. If no job meets the requirements of Rule 2, add J_k to Φ_S.

Step 7: apply Rule 3. Let $k = k-1$. If $k = 1$, go to step 8.

Step 8: final ordering is complete. Apply the original due dates, if any were modified in step 1. Compute total weighted tardiness for the schedule.

Example 2 (see Table 1) demonstrates the application and utility of the three heuristic rules when incorporated in the HMR heuristic.

From Rinnooy Kan et al., J_1 precedes J_3 and is ineligible to occupy position 4. Jobs 2, 3 and 4 are eligible for addition to Φ_S.

$$G_{3,4} = [\min(p_3, T_4)]w_4 = (1)30 = 30,$$

$$CI_{3,4} = [\max(0, p_4 - S_3)]w_3 = (100 - 84)2 = 32,$$

$$NG_{3,4} = 30 - 32 = -2 = OG_3,$$

$$\text{Max } NG_3 = NG_{3,4} = 0,$$

$$G_{2,3} = [\min(p_2, T_3)]w_3 = (0)2 = 0,$$

$$CI_{2,3} = [\max(0, p_3 - S_2)]w_2 = (0)4 = 0,$$

$$NG_{2,3} = 0 = OG_2.$$

Note that the slack of J_B, after relocation to position 3, is reduced by the processing time of J_C to 77.

$$G_{2,4} = [\min(p_2, T_4)]w_4 = (3)30 = 90,$$

$$CI_{2,4} = [\max(0, p_4 - S_2)]w_2 = (100 - 77)4 = 92,$$

$$NG_{2,4} = 90 - 92 = -2 = OG_2.$$

Since neither J_B nor J_C has a positive overall gain, but the eligible set contains two jobs with weights lower than J_D, Rule 2 is invoked. J_B is removed from the schedule and slack and tardiness is recomputed, producing:

Job	p_i	d_i	w_i	S_i	T_i	w_iT_i
A	1	70	3	69	0	0
C	1	89	2	87	0	0
D	100	100	30	0	2	60

J_1 precedes J_2 and is not eligible to occupy position 3. Note also, that the subscripts refer to job position in the current sequence.

$$G_{2,3} = [\min(p_2, T_3)]w_3 = (1)30 = 30,$$

$$CI_{2,3} = [\max(0, p_3 - S_2)]w_2 = (100 - 87)2 = 26,$$

$$NG_{2,3} = 30 - 26 = 4 = OG_2,$$

$$OG_2 + OG_2 = -2 + 4 = 2,$$

Table 1
Example 2: a computational example of the HMR algorithm

Job	p_i	d_i	w_i	S_i	T_i	w_iT_i
A	1	70	3	69	0	0
B	3	82	4	78	0	0
C	1	89	2	84	0	0
D	100	100	30	0	5	150

resulting in the following schedule:

Job	p_i	d_i	w_i	S_i	T_i	w_iT_i
A	1	70	3	69	0	0
D	100	100	30	0	1	30
C	1	89	2	0	13	26
B	3	82	4	0	23	92

Since J_B and J_C are now in the scheduled set, Rule 3 tests the relative positioning of the two jobs.

$$G_{3,4} = [\min(p_3, T_4)]w_4 = (1)4 = 4,$$

$$CI_{3,4} = [\max(0, p_4 - S_3)]w_3 = (3 - 0)2 = 6,$$

$$NG_{3,4} = 4 - 6 = -2$$

and J_C precedes J_B in the scheduled set.

J_D continues to possess tardiness and J_A is now eligible to occupy position 2.

$$G_{1,2} = [\min(p_1, T_2)]w_2 = (1)30 = 30,$$

$$CI_{1,2} = [\max(0, p_2 - S_1)]w_1 = (100 - 69)3 = 93,$$

$$NG_{1,2} = 30 - 93 = -63 = OG_1.$$

J_D is therefore allowed to occupy position 2 and J_A occupies position 1 in the final schedule.

The new heuristic presented in this study (HMR) and the heuristic of Morton and Rachamudugu (MR) were coded in FORTRAN 77 and used to solve problems as described in Section 6. All problems were solved with each algorithm using an IBM 4381 mainframe.

6. Experimental results

Previous computational results have indicated that both the range of processing times and the relative fraction of tardy jobs within a set to be scheduled have a more pronounced effect on problem difficulty than the relative range of due dates. Additionally, the variance in processing times and/or job weights have been shown to have a pronounced effect on problem difficulty. Chambers et al. [3] reported that when both are allowed a fairly wide distribution, even the branch and bound algorithm of Potts and Van Wassenhove [12] did not produce optimal solutions within a 'reasonable' amount of CPU time.

Potts and Van Wassenhove [11] drew integer processing times from the Uniform distribution [1, 100], integer weights from the uniform distribution [1, 10] and integer job due dates from the uniform distribution $P[(1 - t - R/2)]$, $P[(1 - t + R/2)]$, where

$$P = \sum_{i=1}^{n} p_i,$$

t = relative tardiness factor, a parameter specified between 0.0 and 1.0, and R = relative range of due dates, a parameter specified no less than 0.0.

In contrast, Chambers et al. (CCLM) drew job weights from the uniform distribution [1, 10] and the uniform distribution [1, 100], while using the same procedures as Potts and Van Wassenhove (PVW) for selecting processing times and due dates. CCLM confirmed the findings of PVW with respect to the effects of relative range of due dates, R, and the tardiness factor, t. CCLM further showed that increasing the range of job weights, w_i, increased problem difficulty more so than increasing the range of processing time.

In an effort to test competing algorithms under rigorous experimental conditions, we drew job weights and processing times from the uniform distribution [1, 100]. Due dates were established using the PVW method described above and the values, $t = 0.2$, 0.4, 0.6 and 0.8 and $R = 0.2$, 0.4, 0.6, 0.8 and 1.0. Additionally, three levels of problem size were considered: 30, 40 and 50 jobs. Ten replications from each combination of t, R and n-jobs were solved, for a total of 600 problems.

Table 2
Comparison of average weighted tardiness produced by selected heuristics

No. of jobs	Average weighted tardiness			% Difference from HMR	
	HMR	MR	CCLM	MR	CCLM
30	14512	15023	14459	3.52	−0.37
40	14406	15203	14866	5.53	3.19
50	20716	22022	21598	6.30	4.26
All	16325	17203	16805	5.38	2.94

HMR = the new heuristic presented in this study.MR = Morton and Rachamudugu heuristic.CCLM = Chambers, Carraway, Lowe and Morin heuristic.

CCLM benchmarked the results of their heuristic and MR against optimal solutions. The algorithm used to generate optimal solutions, however, was the Schrage-Baker [16] procedure. PVW clearly demonstrated that this algorithm only solves the 'easiest' problems if there are more than 20 jobs in the set to be scheduled. Of the levels used in CCLM, the easiest problems lie in the settings having $t = 0.4$, with $R = 0.4$, 0, 1.2 and $t = 0.8$, with $R = 1.2$. We assume that the results summarized in Table 4 (p. 646) of CCLM are from these cells. Since we have not been able to execute the coding supplied by CCLM, we attempt to provide some sense of comparison by incorporating the results reported in CCLM with our own experimentation that tested the MR and HMR heuristics. Table 2 compares the performance of the MR and HMR algorithms, along with speculated CCLM results. Here, the units of tardiness (from only the settings: $t = 0.4$ with $R = 0.4$, 0.6, 0.8 and 1.0, and $t = 0.8$ with $R = 1.0$) are combined to yield a single measure of performance: weighted-tardiness averaged over all the experimental settings. While CCLM used $R = 1.2$, which is outside our range of testing, we would expect that their use of a wider range of due dates will favor them with lower weighted tardiness values (Table 6 shows the effect of the range of due dates factor). Thus, our speculated comparisons with CCLM are conservatively stated.

The percentage difference for the MR and CCLM heuristics was computed by subtracting the average weighted tardiness produced by the HMR heuristic from the average weighted tardiness produced by the competing heuristic, dividing by the average tardiness produced by the HMR heuristic and converting to a percentage.

The HMR heuristic dominates the MR heuristic in these combined cells. The difference between the HMR heuristic and the CCLM heuristic in the 30-job problems is very slight from a practical perspective. Note that as problem size increases, the HMR heuristic begins to gain a clear advantage. This is explained by the fact that the CCLM algorithm begins with an EDD sequence and uses the Schrage–Baker algorithm to solve the first 15-job sub-problem to optimality. The first five of these jobs are scheduled in the earliest open positions and the next five jobs from the EDD sequence are added to the ten remaining jobs to create a new sub-problem. Solution continues in this fashion until no jobs remain to form a new sub-problem. Small problems (30 or fewer jobs), then, have a fairly high probability of being solved optimally with this technique.

Table 3
Paired t-test results for average weighted tardiness produced by the HMR and MR heuristics

Average weighted tardiness, MR	4384.9
Average weighted tardiness, HMR	4236.9
t computed	4.22
t critical	2.00
p	< 0.0001

Table 4
ANOVA: main-effects of n-jobs, t and R on the percentage difference in average weighted-tardiness between the MR and HMR algorithms

Factor	F	p
n-jobs	0.85	0.434
t	10.72	< 0.001
R	5.17	< 0.001

Both HMR and MR were used to solve exactly the same problems, thus there is a dependence between the two algorithms with respect to weighted tardiness. In this case, a non-parametric technique, such as the paired t-test, eliminates consideration of other variables and allows direct comparison of the effect of solution technique (algorithm) on the performance measure, weighted tardiness (see Ref. [5]). Additionally, no assumptions as to the underlying distributions are necessary (see Ref. [6]). Table 3 presents the results of the paired t-test for all the experimental settings (i.e. $t = 0.2$, 0.4, 0.6 and 0.8; $R = 0.2$, 0.4, 0.6, 0.8 and 1.0).

The reported p-value of less than 0.0001 leads to the conclusion that the HMR algorithm produces significantly less weighted-tardiness than the MR algorithm. Having determined that there is a significant difference in algorithmic performance with respect to weighted tardiness, an analysis of variance (ANOVA) was used next to determine the effect of the independent variables t, n-jobs and R with respect to the percentage difference between the two algorithms. The percentage difference was computed in the following manner:

$$\text{Percent difference} = ((\text{MR} - \text{HMR})/\text{HMR})*100$$

With respect to the percentage difference, the experimental design assures an equal number of random independent observations for each factor. This describes a fixed-effects model for which ANOVA is an appropriate analysis technique (see Ref. [10]). Dividing the desired significance level, 0.05 in this case, by three, the number of experimental factors in the ANOVA model, produces the Bonferroni inequality. Table 4 presents the results of the ANOVA for the experimental factors t, n-jobs and R with respect to percentage difference.

The Bonferroni inequality requires each of these factors to have a p-value less than or equal to 0.0167 in

Table 5
Percent differences in average weighted-tardiness between the MR and HMR algorithms for values of t and R

t	R				
	0.2	0.4	0.6	0.8	1.0
0.2	37.8	60.0	28.2	99.9	0.0
0.4	22.9	38.0	24.2	37.9	23.7
0.6	11.2	21.3	6.2	1.7	2.3
0.8	3.0	0.7	−0.8	−0.5	−0.8

Table 6
Average weighted tardiness produced by the MR and HMR heuristics

Factor	0.2	0.4	0.6	0.8	1.0
t	MR 54	MR 820	MR 4275	MR 12390	not used
	HMR 39	HMR 644	HMR 3932	HMR 12333	not used
R	MR 5890	MR 4684	MR 4357	MR 3701	MR 3283
	HMR 5519	HMR 4287	HMR 4287	HMR 3685	HMR 3281

order to be considered significant at the 5% level. It is thus apparent that both t and R significantly influence relative algorithmic performance. In order to illustrate the effect of these experimental factors, Table 5 shows the relative advantage of HMR over the MR algorithm (expressed as a percentage) when the experimental results are averaged over the various factor levels.

The setting $t = 0.2$, $R = 1.0$, always yields problem sets wherein all jobs can be completed on time. We note that as t increases, the difference in heuristic solutions decreases markedly. This occurs because an increasing fraction of tardy jobs causes an optimal solution to closely approximate a shortest weighted processing time (SWPT) schedule. As previously noted, if no job can be completed early the MR heuristic will create an SWPT schedule.

While the HMR algorithm dominates the MR algorithm when $t = 0.2$, 0.4 and 0.6, the overall difference between the two is only 3.5%. This is due to the large influence of weighted tardiness produced when $t = 0.8$. Table 6 was constructed by summing the units of weighted tardiness produced by a selected algorithm for a given factor (t or R) over all values of the other factor and dividing by the total number of problems. Thus, for the cell $t = 0.2$, the MR heuristic produced an average weighted tardiness of 54 units over all combined values of R and an average weighted tardiness of 5890 units for all values of t when $R = 0.2$. When $t = 0.8$, the average weighted tardiness is equal to

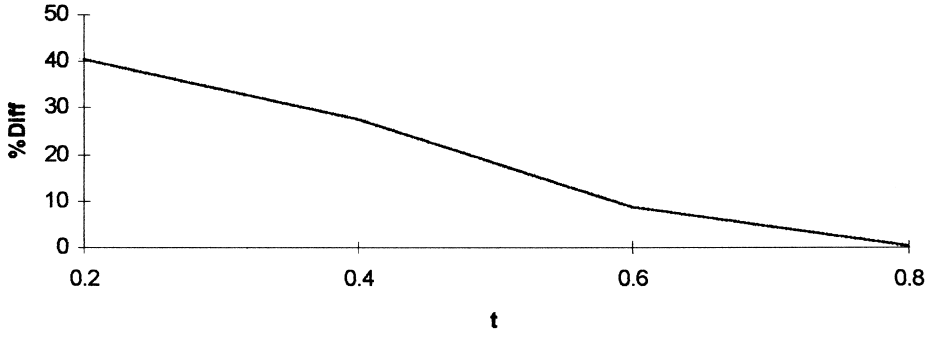

Fig. 1. Percent difference between the MR and HMR heuristics with respect to the relative fraction of tardy jobs.

12333 units, representing nearly 75% of all weighted tardiness for all values of t. The general effect of increasing R, while holding t constant, is to decrease weighted tardiness, but no particular value of R produces a dramatically high percentage of weighted-tardiness.

When more than one experimental factor is considered statistically significant, it is sometimes useful to consider the reported F-value for comparing the relative significance. In this case, the F-value associated with t is twice as large as that associated with R. Table 4 suggests that t has a far greater effect than R with respect to the percentage difference. Fig. 1 depicts the difference in weighted tardiness produced by the two heuristics with respect to t. The clear implication here is that unless a shop has an inordinately high percentage of tardy jobs, the HMR algorithm outperforms MR. Naturally, when a shop suffers high tardiness, managers may wish to consider adding additional machine capacity, beyond the single machine under consideration.

The contention that the HMR algorithm is superior to the MR algorithm is further supported by the fact that, of the 600 problems, HMR found 79.6% 'best' solutions while MR found 38.3% best solutions. A 'best' solution occurs when the solution from a particular heuristic is no worse than that from any other heuristic under consideration. In this way, when both heuristics find the same solution both are credited with a 'best' solution. Additionally, of the 94 problems having no tardiness, MR found an optimal solution 51 times. HMR, of course, *must* find an optimal solution if all jobs can be completed on time, since it begins with an EDD sequence.

Run time, in CPU seconds, is certainly an important consideration when selecting a scheduling method, especially in environments where the method is frequently used. Data on solution speed in CPU seconds collected in this study showed that under the worst case the HMR heuristic required 0.14 seconds to solve a 50-job problem, while the maximum time required for the MR heuristic was 0.03 s. While MR is nearly five times faster, solution speed seems trivial when one considers that more time is required to type in the six-character problem label than to actually solve the problem.

7. Conclusions

Progress in expanding the size of problems that can be solved optimally comes primarily as a result of identifying dominance properties or precedence relationships. These properties reduce the number of sequences to be considered in order to ensure that an optimal solution has been found. The development of Proposition A, along with other work currently underway by the authors, indicates that as yet unidentified precedence relationships for the single-machine, weighted-tardiness problem do exist. This research has pointed the way for development of those properties.

The HMR heuristic is clearly superior to the Morton and Rachamudugu algorithm. Furthermore, the new heuristic seems, by inference, to be superior to the Chambers et al. algorithm, particularly in problems having more than 30 jobs. In shops where penalties due to tardy job completion can be assigned, schedules formed using the HMR heuristic are expected to be superior to schedules using previously reported methods with respect to minimal weighted tardiness.

Appendix A. Proof of Proposition A

Proposition A. An optimal sequence is not altered by considering the due date, $d_{j,}$, of a job, J_j, as the maximum of the job's processing time, $p_{j,}$, or its due date. Thus $dmod_{j,} = \max(d_{j,}, p_j)$ where $dmod_j$ represents a job's due date for sequencing purposes.

Proof. Consider the precedence constraint of Rinnoy Kan et al. [12]. Assume that two jobs, J_j and J_k, are in the set of jobs to be scheduled such that $p_j < p_{k,}$, $w_{j,} \geq w_k$ and $d_j \leq \max[d_k, (p(B_k) + p_k)]$, where

$$p(B_k) = \sum_{i \in B_k} p_i \tag{A.1}$$

Under these conditions, J_j will precede J_k in at least one optimal sequence. From Eq. (A.1), d_j and d_k can both have values less than zero. The terms $p(B_k)$ and p_k must be strictly non-negative since job processing times are defined as positive integers. The bracketed term in Eq. (A.1) must always assume a value greater than or equal to p_k. Thus, in considering job due dates for sequencing purposes we will never consider a value less than the job's processing time.□

References

[1] Baker KR, Schrage L. Finding an optimal sequence by dynamic programming: an extension to precedence-related tasks. Oper. Res. 1978;26:111–20.

[2] Blackburn JD, Millen RA. Heuristic lot sizing performance in a rolling schedule environment. Decision Sci. 1980;11(4):691–701.

[3] Chambers RJ, Carraway RL, Lowe TJ, Morin TL. Dominance and decomposition heuristics for single machine scheduling. Oper. Res. 1991;39:639–47.

[4] Elmaghraby SE. The one-machine sequencing problem with delay costs. J. Indust. Eng. 1968;XIX:105–8.
[5] Gruenther WC. Analysis of variance. Englewood Cliffs, NJ: Prentice-Hall, 1968.
[6] Hays WL. Statistics. New York, NY: Holt, Rinehart and Winston, 1988.
[7] Lawler EL. A pseudo-polynomial algorithm for sequencing jobs to minimize total tardiness. Ann. Discrete Math. 1977;1:331–42.
[8] McNaughton R. Scheduling with deadlines and loss functions. Management Sci 1959;6(1):1–12.
[9] Morton TE, Rachamudugu RMU. Myopic heuristics for the single machine weighted tardiness problem. Working Paper 30-82-83. Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, 1982.
[10] Neter J, Wasserman W, Kutner MH. Applied linear statistical models. Homewood, IL: Irwin, 1985.
[11] Potts CN, VanWassenhove L. A decomposition algorithm for the single machine, total tardiness problem. Oper. Res. Lett. 1982;1:177–81.
[12] Potts CN, VanWassenhove L. A branch-bound algorithm for the single machine, total weighted tardiness problem. Oper. Res. 1985;33:363–77.
[13] Rinnooy Kan AHG, Lageweg BJ, Lenstra JK. Minimizing total costs in one-machine scheduling. Oper. Res. 1975;23:908–27.
[14] Russell RM, Holsenback JE. A comparative evaluation of a tardiness minimizing heuristic in a general job shop: an extended abstract. In: Sumichrast RT, coordinator. National DSI Conference Proceedings, San Francisco, CA. 1993.
[15] Schild A, Fredman IJ. On scheduling tasks with associated linear loss functions. Manage. Sci. 1961;7:280–5.
[16] Schrage L, Baker KR. Dynamic programming solutions of sequencing problems with precedence constraints. Oper. Res. 1978;26:444–9.
[17] Shwimer J. On the *N*-job, one-machine, sequence-independent scheduling problem with tardiness penalties. Manage. Sci. 1972;18:B301–B313.
[18] Wilkerson LJ, Irwin JD. An improved method for scheduling tasks. AIIE Trans. 1971;3:239–45.