ELSEVIER

European Journal of Operational Research 145 (2003) 45–56

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Discrete Optimization

Minimizing the weighted number of tardy jobs on a single machine

Rym M'Hallah [a], R.L. Bulfin [b,*]

[a] *Department of Quantitative Methods, Institut Superieur de Gestion de Sousse, B.P. 763, Sousse 4000, Tunisia*
[b] *Department of Industrial and Systems Engineering, Auburn University, 207 Dunstan Hall, Auburn, AL 36849-5346, USA*

Received 8 November 2000; accepted 12 November 2001

Abstract

In this paper, we describe an exact algorithm to solve the single machine weighted number of tardy jobs scheduling problem. The algorithm uses branch-and-bound with the bounds obtained from a surrogate knapsack. Extensive computational experiments are carried out. These experiments validate previous work on what constitutes a difficult instance and show that even difficult instances with 2500 jobs can be solved optimally in a reasonable amount of time.

Keywords: Scheduling; Combinatorial optimization

1. Introduction

Scheduling problems are frequently encountered in actual industrial settings, such as transportation, logistics, telecommunication, media management, sport, reliability, and manufacturing. Some scheduling problems are exceptionally difficult to solve. The inherent combinatorial explosiveness caused by the underlying sequencing problem requires that special approaches be designed to exploit the particular structure of each problem.

The single machine scheduling problem has been well studied. Many variations of measures of performance, job characteristics, precedence and release times provide a variety of single machine models. In this paper, we address the problem of scheduling a single machine to minimize the weighted number of tardy jobs. There are n jobs to be processed on a single machine, each having a processing time, weight and due date. The optimal solution maximizes the total weight of on-time jobs. Consider a large engineering shop that places bids on jobs, and has only one person preparing the bids. Here, the processing time is how long it takes to prepare a bid, the job weight is the expected profit if the bid is accepted and the due date is the bid deadline. If the bid cannot be prepared by the deadline, there is no reason to prepare it, since it will accrue no profit.

Minimizing the number of tardy jobs on a single machine is an easy problem to solve (Moore, 1968). However, if all jobs are not equally

* Corresponding author. Tel.: +1-334-844-1422, fax: +1-334-844-1381.
E-mail addresses: Rym.Mhallah@irsit.rnrt.tn (R. M'Hallah), bulfin@eng.auburn.edu (R.L. Bulfin).

0377-2217/03/$ - see front matter
PII: S0377-2217(02)00180-7

important, i.e., have different weights, the problem is *NP-Hard*, even if all jobs have a common due date (Karp, 1972). Several heuristic and enumerative algorithms have been proposed for the problem, but exact algorithms for large problems (e.g., more than 1000 jobs) do not exist. We develop both a heuristic and an exact algorithm capable of solving problems with 2500 jobs.

This paper is organized as follows. Section 2 gives a brief overview of previous work on the problem. Section 3 contains a mathematical programming formulation used to develop a surrogate bound based on the knapsack problem, followed by a formal statement of the algorithm. Section 4 describes and analyzes computational experiments. Finally, Section 5 is a summary.

2. Literature review

The single machine weighted number of tardy jobs problem has received considerable attention. Proposed exact methods are based on dynamic programming and branch-and-bound. Dynamic programming formulations, such as those provided by Lawler and Moore (1969) and Sahni (1976), cannot practically solve even small problems. Indeed, both algorithms require computational times and memory storage that are highly dependent on the input data.

Branch-and-bound procedures on the other hand have been computationally more attractive. For instance, Villarreal and Bulfin (1983) introduced new branch-and-bound algorithms that solved 50-job problems in a few seconds of computer time. Their algorithm was based upon a dominance theorem and lower bounding procedures. The dominance theorem they developed is very useful for reducing the size of the problem; thus speeding the search for optimal solutions. Their bounds are based on solving unit processing time or unit weight problems. Tang (1990) proposed a branch-and-bound algorithm that handles up to 85 jobs. His algorithm employed some newly introduced dominance rules between jobs.

Potts and Van Wassenhove (1988) suggested several versions of a branch-and-bound algorithm. They used the Villarreal and Bulfin (1983) dominance theorem to derive a reduction. In addition, they derived lower bounds based on applying Lawler and Moore's dynamic programming approach to scaled versions of the problem and on a linear programming relaxation of the zero–one integer programming formulation. The success of the reduction algorithm and the particular coding they adopted for the dynamic programming storage enabled them to solve instances with up to 1000 jobs. Finally, Dauzere Peres (1995) considered the more general problem with release dates. He proposed an exact branch-and-bound method using dominance rules and two lower bounds to solve 140-job problems.

3. Background

Scheduling researchers have long been aware minimizing the weighted number of tardy jobs on a single machine can be modeled as an integer program. For ease of discussion, we will present the model as maximizing the weighted number of on-time jobs, an equivalent problem.

Let p_j be the processing time of job j, w_j the weight of job j, d_j the due date of job j, and x_j be one if job j is on time and zero otherwise.

Further, let the jobs be numbered in earliest due date (EDD) order, i.e., $d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n$. The mathematical model of the weighted number of tardy jobs problem (WNT) is

(WNT)

$$\begin{aligned} \max \quad & \sum_{j=1,n} w_j x_j \\ \text{s.t.} \quad & \sum_{i=1,j} p_i x_i \leqslant d_j \quad j = 1, \ldots, n, \\ & x_j \in \{0,1\} \quad j = 1, \ldots, n. \end{aligned}$$

The objective simply sums the weights of all on-time jobs. The constraints use the fact that if any sequence has all jobs on time, the EDD sequence does (Jackson, 1955). The left-hand side of constraint j is the completion time of job j if it is on time since all on time jobs with smaller due dates will be scheduled before it. If job j is to be on time, its completion time must be no greater than its

due date. The final set of constraints, the variables are binary, force a job to be either on time or tardy.

Surrogate constraints have long been used in integer programming to provide bounds (Parker and Rardin, 1988). Let λ_i be a surrogate multiplier associated with constraint i. Using λ_i, we can combine the n constraints of WNT into a single constraint. Define

$$a_j = p_j \sum_{i=j,n} \lambda_i, \quad \text{and}$$

$$b = \sum_{j=1,n} \lambda_j d_j.$$

For any $\lambda_i \geqslant 0$, the following knapsack model (KP) provides a bound on the solution of WNT:

(KP)

$$\max \quad \sum_{j=1,n} w_j x_j$$

$$\text{s.t.} \quad \sum_{j=1,n} a_j x_j \leqslant b,$$

$$x_j \in \{0,1\}, \quad j = 1,\ldots,n,$$

since any feasible solution to WNT is also feasible to KP, but not vice versa.

When using (KP) as a relaxation, the quality of the bound is determined by the surrogate multipliers. The tightest bound is obtained by solving the surrogate dual (SD):

(SD)

$$\min_{\lambda_i \geqslant 0} \max \quad \sum_{j=1,n} w_j x_j$$

$$\text{s.t.} \quad \sum_{j=1,n} \left(p_j \sum_{i=j,n} \lambda_i \right) x_j \leqslant b,$$

$$x_j \in \{0,1\}, \quad j = 1,\ldots,n.$$

The bound obtained from SD is no worse than bounds obtained from linear programming or Lagrangean relaxations of WNT. Methods for finding the surrogate multipliers by solving SD often converge slowly and sometimes are unstable. As an approximation, we use multipliers found by subgradient optimization (Parker and Rardin, 1988).

The linear programming relaxation of KP (LPKP) is obtained by replacing the binary restrictions with $0 \leqslant x_j \leqslant 1$, $j = 1,2,\ldots,n$. Its solution is easy to calculate. Reorder the variables so that

$$w_1/a_1 \geqslant w_2/a_2 \geqslant \cdots \geqslant w_n/a_n.$$

Let f be the index such that

$$\sum_{i=1,f} a_i \leqslant b \leqslant \sum_{i=1,f+1} a_i.$$

Then the optimal solution to LPKP is

$$x_j = 1, \quad j = 1,2,\ldots,f-1,$$

$$x_j = \left(b - \sum_{i=1,f} a_i \right) \Big/ a_f, \quad j = f,$$

$$x_j = 0, \quad j = f+1, f+2, \ldots, n.$$

The optimal solution will have at most one fractional valued variable. Variables with ratios smaller than the ratio of the fractional valued variable will be zero, while those with larger ratios will be one. We can use this as the basis of a heuristic.

3.1. A heuristic for WNT

A heuristic procedure for WNT is helpful as an initial solution for an exact algorithm and is used in the subgradient search. Also, for extremely large problems, a heuristic may be the only way to get a solution. We propose a heuristic that is an extension of the "slippery algorithm" for KP (Woolsey and Swanson, 1975). It starts with the solution of LPKP and sets variables to 0 until a feasible solution to WNT is found. The order in which the variables are set to 0 is based on the ratio of objective to constraint coefficients. Once a feasible solution to WNT is found, we try to improve it. This is done by temporarily setting variables with value 0 to 1 and testing feasibility; if the resulting solution is infeasible, the variable is returned to its 0 value. Again, the order in which the variables are examined is based on the ratios. Formally, the heuristic is:

Step 0: Solve LPKP. Set $E = \{1,2,\ldots,f-1\}$ $T = \{f, f+1, \ldots, n\}$ and $k = f-1$.

Step 1: If the jobs represented by the set E are all on time in EDD order, go to Step 3.

Step 2: Set $E = E - \{k\}$, $T = T \cup \{k\}$ and $k = k - 1$ and go to Step 1.

Step 3: If $T = \emptyset$ go to Step 5; otherwise, set $k = k + 1$, $T = T - \{k\}$.

Step 4: If the jobs represented by the set $E \cup \{k\}$ are all on time in EDD order, set $E = E \cup \{k\}$. Go to Step 3.

Step 5: The heuristic solution to WNT has all jobs in E on time with objective value

$$z^* = \sum_{j \in E} w_j, x_j^* = 1 \quad j \in E, \text{ otherwise } x_j^* = 0.$$

Step 0 requires sorting the ratios so its complexity is $O(n \ln n)$. Steps 1 and 2 can be executed at most n times and Step 1 needs at most n operations (assuming the jobs are already sorted in EDD order), so these steps have complexity $O(n^2)$. Steps 3 and 4 are equivalent to Steps 1 and 2, so the complexity is $O(n^2)$. Therefore, the complexity of the heuristic is $O(n^2)$. This does not include calculating the surrogate multipliers. If at least one job has $p_j \leqslant d_j$, the heuristic will always find a feasible solution. If $p_j > d_j$, $j = 1, n$, all jobs are tardy regardless of their sequence; thus, no feasible solution exists. We do not consider these trivial cases.

3.2. An exact algorithm for WNT

The best solution to KP that is feasible to WNT, say x^*, is the optimal solution to WNT. If not, there exists a feasible solution to WNT, say x^+, with $wx^+ > wx^*$. But because KP is a surrogate relaxation of WNT, any feasible solution to WNT is also feasible to KP, so this contradicts the fact that x^* is the best solution to KP that is feasible to WNT.

A modification of a branch-and-bound algorithm for KP can be used to solve WNT. It uses bounds based on LPKP and when a feasible integer solution to LPKP is found, it is checked for feasibility in WNT. If it is feasible, we can fathom the candidate problem; further, if this feasible solution is better than the incumbent solution, it replaces the incumbent solution. If it is not feasible to WNT, rather than fathoming the node, additional variables must be fixed and the procedure continues. A statement of the algorithm follows.

Step 0: Apply the heuristic to WNT; let z^* be the solution value and x_j^* the value of the variables. Initialize the candidate list to consist of WNT, and let all variables be free.

Step 1: If the candidate list is empty, stop z^* and x_j^* solve WNT. Otherwise choose the problem on the candidate list with the best bound; denote the problem by CP.

Step 2: Solve CPLP, the linear programming relaxation of CP, using the ratio algorithm. Let the solution value be z with variable values x_j.

Step 3: If CPLP has no feasible solution or $z \leqslant z^*$, go to Step 1.

Step 4: If some x_j is not integer, go to Step 5. If the jobs represented by $x_j = 1$ are all on time in EDD order, go to Step 7.

Step 5: Let z_1 be the solution to CPLP with the free variable having the largest ratio of objective to constraint coefficient changed to a value of zero. Similarly, let z_2 be the solution to CPLP with the free variable having the smallest ratio of objective to constraint coefficient changed to a value of one. If $z_1 < z_2$ let j^+ be the index of the variable with the largest ratio; otherwise, let j^+ be the index of the variable with the smallest ratio.

Step 6: Separate the candidate problem into two new candidate problems. The first one consists of CP with the constraint $x_{j^+} = 1$ added. The second is CP with $x_{j^+} = 0$ added. These variables are no longer free, but are fixed at the stated value. Place both new problems on the candidate list and go to Step 1.

Step 7: A new incumbent has been found. Set $z^* = z$ and $x_j^* = x_j$, $j = 1, 2, \ldots, n$. Remove all problems from the candidate list with bounds no better than the new z^* and go to Step 1.

Neither of the algorithms takes advantage of the dominance/reduction properties, since we wanted to keep the algorithms as simple as possible. These algorithms can be implemented with slight modification to any existing knapsack code, i.e., add a feasibility check for WNT when an integer solution is found, and a standard subgradient search routine. No attempt was made to optimize the code.

4. Computational experiments

The preceding algorithm was extensively tested on a battery of randomly generated problems. The purpose of this experiment was twofold: to examine the performance of the algorithm, and to investigate the possible influence of certain problem parameters on the difficulty of the problems. We follow the experimental procedure of Villarreal and Bulfin (1983), since it appears to be the most comprehensive study.

4.1. Problem generation

A test instance can be considered as a sample of size n, where the jth observation (p_j, d_j, w_j) corresponds to the processing time, due date and weight of job j. Due dates were drawn from a uniform distribution with mean μ_d and range R_d. Processing times and weights were assumed to come from a bivariate normal distribution restricted to the positive quadrant by ignoring negative values. The mean and standard deviation for processing times and weights are μ_p, σ_p and μ_w, σ_w, respectively. The correlation coefficient for the joint distribution is ρ_{pw}. Further, we require $p_j \leqslant d_j \leqslant \sum_{i=1,n} p_i$, $j = 1, 2, \ldots, n$, to prohibit trivial cases in the data.

Under these assumptions, eight parameters are needed to specify problems: n, μ_p, σ_p, μ_w, σ_w, ρ_{pw}, μ_d and R_d. Since the magnitude of the processing times and weights are typically not critical to branch-and-bound algorithms, μ_p was fixed at 100 and μ_w at 20. Of the five other parameters, or factors, to vary in the experiment, only the correlation coefficient was specified directly. The remaining four were specified in terms of four surrogate factors. The variances of the processing times and weights were expressed through their coefficients of variation. The mean and range of the due dates were expressed in terms of the relative range of the due dates and the tardiness factor. These surrogate factors are dimensionless quantities and the latter two are often discussed in studies of scheduling problems with due dates. They allow the due dates to be related to the expected makespan in a controllable way.

Each factor was tested at the two levels proposed by Villarreal and Bulfin (1983). They claimed the lower level produced instances that are relatively easy to solve while the higher level instances were relatively difficult. They number factors in the order of influence, i.e., factor one had more impact on difficulty than factor two, etc. for their algorithm. We give factors and their levels here; details can be found in Villarreal and Bulfin (1983).

Relative range of due dates; $f1 = 1 - R_d/(n * \mu_p)$; {0.30, 0.80}.

Correlation between processing times and weights; $f2 = \rho_{pw}$; {−0.70, 0.0}.

Tardiness factor; $f3 = 1 - \mu_d/(n * \mu_p)$; {0.40, 0.65}.

Relative variation of processing times; $f4 = \sigma_p/\mu_p$; {0.20, 0.80}.

Relative variation of job weights; $f5 = \sigma_w/\mu_w$; {0.20, 0.80}.

For each of the 32 combinations of problem parameters, 10 instances were randomly generated from the appropriate distributions. All data were rounded off to the nearest integer. A label, consisting of five binary digits, identifies each problem set. Since each factor takes on only two values, the ith digit will be one if factor i is set at its high level, and 0 if it is at its low level. Therefore, each problem set is completely specified by its label. The results of this experiment, as well as some additional computational runs, are discussed in the following section.

4.2. Computational results

Run time and heuristic quality of the exact and heuristic algorithms were tested for five different job sizes: 500, 1000, 1500, 2000, and 2500. All problems were solved on a Pentium III 733 MHz processor with 128 MB of memory. Ten replications of each of the 32 problem sets were run for each problem size. For each set, the mean, median, and maximum solution times for the subgradient algorithm, heuristic algorithm, exact algorithm and also for solution qualities are tallied. The mean reflects the aggregate of the class while the median provides information about 'typical' instances and is not affected by outliers. Finally, the maximum of each set is an indicator of 'difficult' instances.

After running all 1600 problems, there was little difference in the mean and median for any set of parameters, so we report only the mean and maximum in most cases. Also, the run time for both the subgradient and heuristic algorithms did not appear to vary with the particular levels of the factors, so we do not give data for each distinct problem set. Storage was not a problem for any instance. Fig. 1 gives the run time, in CPU seconds, of the three algorithms plotted against the number of jobs. Since the heuristic algorithm is, in the worst case, $O(n^2)$, there is little growth in average time as the number of jobs increases and takes negligible time, even for the 2500-job instances. As expected, the exact algorithm appears to exhibit exponential growth; 2500-job problems seem to be approaching the limit. The time to execute the subgradient algorithm also grows slowly (low order polynomial due to a limit on the number of iterations), as the number of jobs increases.

Table 1 gives the mean and maximum run time (in CPU seconds) for the branch-and-bound algorithm for each of the 32 problem sets with 500, 1000, 1500, 2000 and 2500 jobs. From these results, optimal solutions for even the hardest 1000-job instances can be found in about 1 minute and hardest 2500-job problems in less than 12 minutes on a desktop computer. As we would expect of combinatorial problems, we see from the maximum and mean in the table that often there is one problem in the set of 10 that is considerably harder than the others. Table 1 has a summary over all instances of the run time of the exact algorithm for each problem size. This summary includes the minimum and median times over all 320 problems of each size. As in Fig. 1, Table 1 indicates the run time for the exact algorithm grows exponentially in the number of jobs n.

Table 2 is similar to Table 1, but examines the heuristic quality. To quantify the heuristic quality, the ratio Zopt/Zheu is computed, where Zopt is the optimal solution and Zheu is the heuristic solution. For all sets of problems, the median of Zopt/Zheu = 1, indicating for many instances, the proposed heuristic generates the optimal solution. As was the case for run time, there is often one problem in the set with quality much worse than the others. This is illustrated by the means in the summary being close to one, while the maximum can be very large. This table indicates that for the vast majority of problems, the heuristic finds optimal or very near optimal solutions. However, when it does not, the large difference justifies the added time needed to find the optimal solution. Fig. 2 shows heuristic quality versus the number of jobs. It illustrates that as the problems get larger, the quality decreases and is particularly noticeable for the maximum of the very large (2000 and 2500 job) problems.

Table 1 indicates that instances with some factors at the high level are harder, and in general, the more factors at the high level the harder the problem class. Table 2 shows the quality of the heuristic algorithm also depends on the difficulty of the particular instance of the problem class. To investigate the effect of the factors on solution

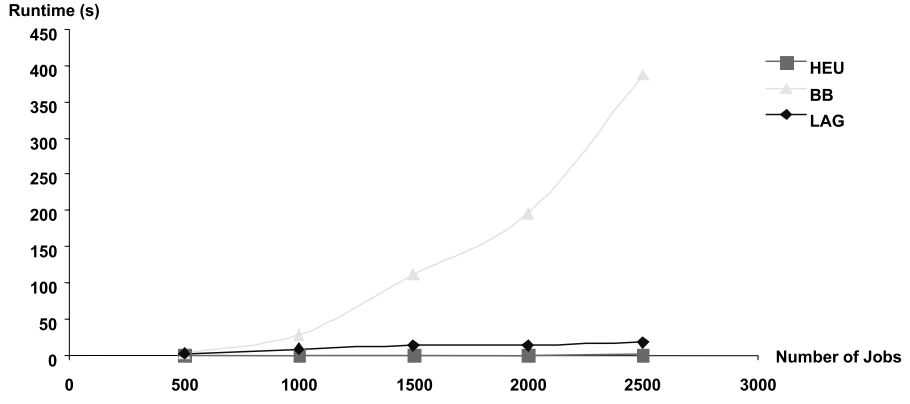

Fig. 1. Algorithm's runtime versus number of jobs.

Table 1
CPU seconds for the exact algorithm

Problem parameters	$n = 500$		$n = 1000$		$n = 1500$		$n = 2000$		$n = 2500$	
	Mean	Max.	Mean	Max.	Mean	Max.	Mean	Max.	Mean	Max.
00000	2.74	2.85	27.46	39.28	92.21	141.94	162.68	168.91	328.78	345.42
00001	2.88	4.56	20.33	31.19	81.63	120.95	161.01	163.09	340.86	434.02
00010	2.69	2.87	31.28	39.82	75.33	134.03	192.04	241.85	377.33	421.01
00011	2.92	5.22	33.05	39.17	84.18	98.33	164.73	174.77	322.98	337.91
00100	1.96	3.69	31.42	65.59	150.79	387.07	181.16	194.10	402.49	509.71
00101	1.43	5.05	29.34	41.36	107.07	153.85	194.17	241.24	397.79	535.97
00110	2.36	3.18	28.27	60.31	152.57	223.61	174.78	194.71	347.56	409.37
00111	1.25	3.46	33.08	39.94	123.27	172.25	196.21	259.48	409.12	562.39
01000	2.68	2.76	23.28	39.87	95.56	160.06	167.04	175.61	332.42	344.17
01001	2.84	4.56	22.86	40.11	80.10	83.54	164.56	168.08	355.10	434.64
01010	2.68	2.80	25.92	39.82	103.81	160.72	180.48	190.11	392.58	488.13
01011	2.62	2.81	24.08	39.43	88.05	167.20	170.22	237.92	394.89	411.73
01100	3.22	4.51	38.14	59.82	116.47	181.85	211.25	290.24	432.34	640.81
01101	1.86	6.65	31.49	41.63	125.83	189.77	172.50	207.51	421.12	544.00
01110	3.85	5.50	27.31	39.38	110.43	182.09	183.10	198.56	414.56	662.89
01111	4.15	7.36	27.23	39.39	99.65	161.27	194.36	267.66	349.45	414.46
10000	5.05	5.93	30.63	51.74	101.30	173.35	186.56	201.36	395.06	404.10
10001	4.76	5.88	30.58	52.63	112.17	206.26	193.71	234.32	377.48	398.81
10010	3.00	3.91	29.25	49.99	144.52	441.49	195.33	211.79	405.60	411.06
10011	3.16	3.90	32.10	48.55	94.88	177.08	181.45	194.93	381.01	422.27
10100	3.53	4.95	30.40	46.08	169.71	259.58	236.64	252.87	400.00	432.26
10101	2.83	3.36	28.20	51.47	107.73	168.18	220.18	248.55	445.00	466.48
10110	2.82	3.20	31.17	45.59	151.65	223.78	240.26	331.15	444.59	473.73
10111	2.84	3.25	22.59	26.26	102.93	164.67	202.54	228.54	417.78	435.73
11000	2.99	3.91	26.29	30.26	124.83	175.38	198.60	204.28	344.72	388.05
11001	2.84	3.36	28.49	50.97	89.15	147.25	193.54	208.66	367.61	384.86
11010	2.91	3.29	26.37	44.28	111.94	179.11	200.77	280.29	363.04	383.88
11011	2.94	3.79	21.67	24.11	92.71	127.48	197.06	227.01	368.36	398.06
11100	2.91	3.18	24.03	42.90	135.68	205.71	237.59	243.94	387.11	427.66
11101	2.71	2.80	27.38	42.03	110.66	249.80	232.60	235.58	430.58	437.76
11110	2.82	3.15	26.81	40.92	128.94	195.97	239.87	313.91	409.29	441.98
11111	2.69	2.81	25.33	42.08	98.73	124.96	233.53	257.17	440.47	452.26
Summary										
Minimum	0.00		6.16		55.59		51.70		163.06	
Median	2.76		20.55		92.5		192.63		388.72	
Mean	2.90		27.99		111.39		195.64		387.41	
Maximum	7.36		65.59		441.49		331.15		662.89	

times, an analysis of variance (ANOVA) was carried out.

The ANOVA considered the factors $f1, f2, f3, f4$ and $f5$ at low and high levels and n at five levels, 500, 1000, 1500, 2000 and 2500. As expected, n was significant at the 0.1% level. The $n * f1$, $n * f3$, $n * f5$, $n * f1 * f2$ and $n * f1 * f3$ interactions were also significant at the 0.1% level. A few other interactions with n were significant at the 10% level, but most were not significant even at the 25% level.

As the relative range of due dates, $f1$, increases from its low level to its high level, the computation time increases. The ANOVA test shows $f1$ is

Table 2
Heuristic quality: Zopt/Zheu

Problem parameters	$n = 500$		$n = 1000$		$n = 1500$		$n = 2000$		$n = 2500$	
	Mean	Max.	Mean	Max.	Mean	Max.	Mean	Max.	Mean	Max.
00000	1.00	1.00	1.00	1.00	1.00	1.00	1.06	1.30	1.00	1.00
00001	1.00	1.00	1.16	1.74	1.16	1.74	1.01	1.05	1.16	1.74
00010	1.00	1.00	1.12	1.71	1.00	1.00	1.02	1.16	1.12	1.71
00011	1.00	1.00	1.12	1.39	1.14	2.37	1.00	1.00	1.12	1.39
00100	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
00101	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.00	1.00
00110	1.00	1.01	1.00	1.02	1.00	1.02	1.01	1.03	1.00	1.02
00111	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.08	1.00	1.00
01000	1.00	1.00	1.00	1.00	1.00	1.00	1.02	1.19	1.00	1.00
01001	1.00	1.00	1.09	1.22	1.00	1.00	1.00	1.04	1.09	1.22
01010	1.00	1.00	1.13	1.65	1.22	2.64	1.00	1.04	1.13	1.65
01011	1.47	5.67	1.11	1.50	1.00	1.00	1.00	1.00	1.11	1.50
01100	1.00	1.02	1.01	1.07	1.00	1.00	1.00	1.00	1.01	1.07
01101	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.00	1.00
01110	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
01111	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.00	1.00
10000	1.00	1.00	1.01	1.09	1.00	1.00	7.22	63.08	1.01	1.09
10001	1.00	1.00	1.00	1.00	4.37	19.58	1.01	1.09	40.69	397.88
10010	1.00	1.00	1.00	1.00	1.00	1.00	7.20	63.00	1.00	1.00
10011	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.05	1.00	1.00
10100	1.00	1.00	1.00	1.00	1.00	1.00	1.02	1.02	1.00	1.00
10101	1.00	1.00	1.00	1.00	1.00	1.00	1.02	1.04	1.00	1.00
10110	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.02	1.00	1.00
10111	1.04	1.40	1.46	3.45	1.00	1.00	1.13	2.19	1.46	3.45
11000	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.04	1.00	1.00
11001	1.04	1.20	1.24	3.20	1.00	1.00	1.00	1.00	1.24	3.20
11010	1.05	1.54	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
11011	1.06	1.23	1.19	2.40	1.08	1.66	1.57	3.86	1.19	2.40
11100	1.00	1.00	1.93	10.31	1.00	1.00	1.01	1.02	1.93	10.31
11101	3.00	15.65	1.20	3.04	1.38	2.89	1.04	1.05	1.20	3.04
11110	1.21	3.07	1.00	1.00	1.00	1.00	1.00	1.01	1.00	1.00
11111	1.91	9.18	1.00	1.00	1.73	6.87	1.03	1.04	1.00	1.00
Summary										
Minimum	1.00		1.00		1.00		1.00		1.00	
Median	1.00		1.00		1.00		1.00		1.00	
Mean	1.12		1.09		1.16		1.42		2.33	
Maximum	15.65		10.31		19.58		63.08		397.88	

significant at the 0.1% level. This deterioration becomes more pronounced as the number of jobs increases due to the $n * f1$ interaction. This is in line with the results of Villarreal and Bulfin (1983) as well as studies for other due-date related scheduling problems. This appears true for the quality of the heuristic solution, as demonstrated by Fig. 3.

Counter to the results of Villarreal and Bulfin (1983), instances with correlation between weights and processing times, $f2$, at its "harder" level are no harder to solve than instances with $f2$ at its "easier" level. This factor was not significant in the ANOVA, even at the 48% level. The $f1 * f2$ interaction was significant at the 0.1% level however.

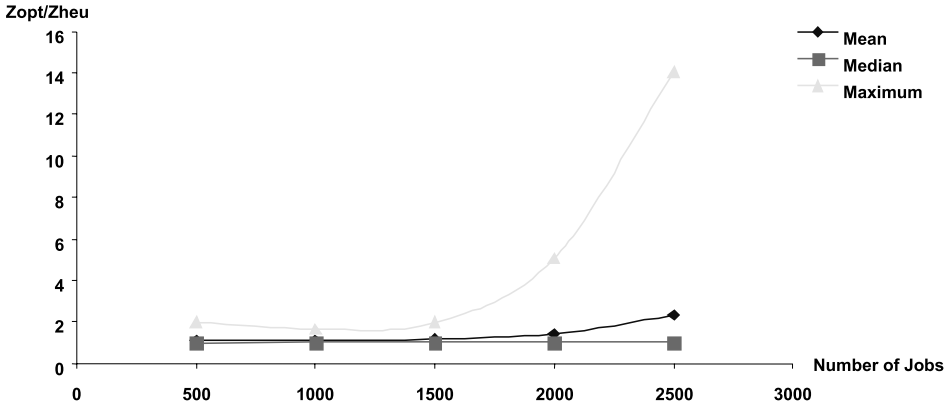

Fig. 2. Heuristic quality versus number of jobs.

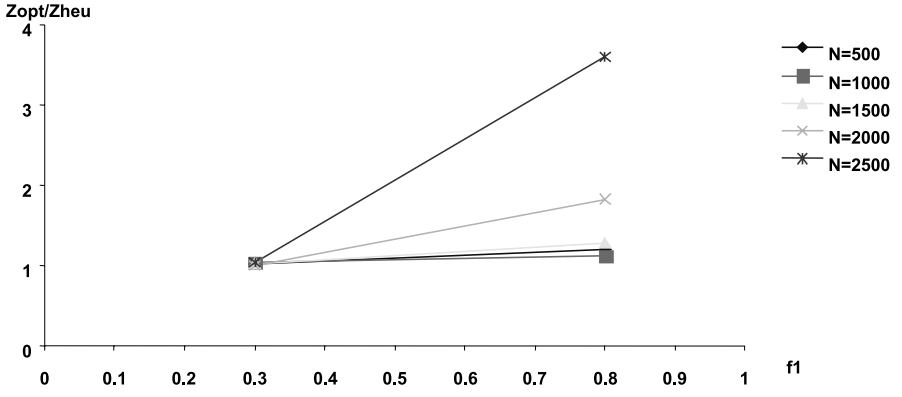

Fig. 3. Heuristic quality versus $f1$.

To further investigate this, additional problem sets with positive correlation between processing times and weights were solved for 1000-job instances. For our algorithm, negatively correlated instances are somewhat harder to solve than positively correlated or uncorrelated ones. This is a consequence of the bounds used in the two approaches. The bounds of Villarreal and Bulfin (1983) are based on either an equal processing time or an equal weight relaxation, so it is not surprising that correlation would adversely affect their bounds, causing their algorithm to take longer.

The tardiness factor, $f3$, is significant at the 0.1% level. To better understand this factor, additional instances were solved for $f3 = 0.5$ and 0.6. The results illustrated in Fig. 4 show that the instances with a tardiness factor around 0.5 are the easiest to solve and that the instances get harder as $f3$ is further away from 0.5. As $f3$ approaches zero or one, we would expect them to become easier.

The relative variation of processing times, $f4$, was not significant at the 61% level, so it does not directly contribute to problem difficulty. However, the $f3 * f4$ interaction is significant at the 0.1% level, so $f4$ indirectly affects problem difficulty.

Finally, the ANOVA shows the relative variation of job weights, $f5$, is significant at the 0.1% level. Thus, when job weights have more variance, the instances are harder to solve. The $f4 * f5$ interaction is significant at the 7% level.

Because n has such a strong influence on the ANOVA, we also did five separate ANOVAs for each level of n. The conclusions on $f1$ through $f5$ were essentially the same as previously discussed.

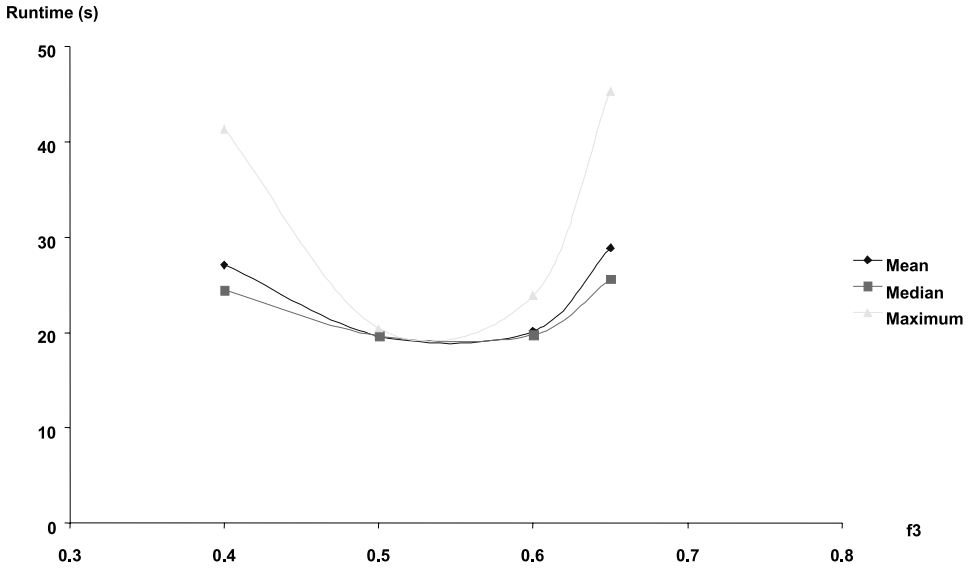

Fig. 4. Exact runtime versus $f3$ (1000 jobs).

Additional tests, following Potts and Van Wassenhove (1988), were conducted for 1000-job problems with large coefficients. For each job j, an integer processing time p_j and an integer weight w_j are drawn from a uniform distribution $[1, a]$. We set $a = 1000$ for the first set of problems and $a = 10^6$ for the second set. An integer due date d_j is drawn from a uniform distribution $[Pd^1, Pd^u]$,

Table 3
CPU seconds for the exact algorithm (PVW problems)

Problem parameters	$a = 10^3$		$a = 10^6$	
	Mean	Max.	Mean	Max.
11	30.76	67.83	16.29	46.42
12	30.81	38.34	28.65	36.37
13	29.95	46.81	26.10	38.95
14	34.75	60.53	31.03	50.20
15	19.57	20.33	13.87	22.52
22	27.51	40.27	30.94	73.67
23	27.24	45.15	23.43	46.81
24	30.62	42.03	28.81	40.05
25	19.34	19.51	15.45	19.44
33	31.14	39.22	29.69	42.89
34	31.11	40.00	29.07	40.11
35	19.33	19.40	19.30	19.34
44	29.26	40.04	29.98	40.10
45	19.32	19.34	19.31	19.38
Summary				
Minimum	0.00		0.00	
Median	24.50		21.58	
Mean	27.19		24.42	
Maximum	67.83		73.67	

Table 4
Heuristic quality (PVW problems)

Problem parameters	$a = 10^3$		$a = 10^6$	
	Mean	Max.	Mean	Max.
11	1.01	1.06	1.01	1.05
12	1.02	1.03	1.49	3.10
13	1.00	1.01	1.87	4.28
14	1.00	1.00	1.30	3.94
15	1.00	1.00	1.00	1.00
22	1.01	1.03	1.01	1.02
23	1.00	1.00	1.46	4.01
24	1.00	1.01	1.00	1.00
25	1.00	1.00	1.00	1.00
33	1.00	1.00	1.06	1.60
34	1.00	1.00	1.00	1.00
35	1.00	1.00	1.00	1.00
44	1.00	1.02	1.00	1.00
45	1.00	1.00	1.00	1.00
Summary				
Minimum	1.00		1.00	
Median	1.00		1.00	
Mean	1.00		1.16	
Maximum	1.06		4.28	

where P is the sum of the processing times of all jobs, and $d^l = 0.2$, 0.4, 0.6, and 0.8, and $d^u = 0.2$, 0.4, 0.6, 0.8, and 1.0. Ten problems were generated for each of the 14 pairs of d^l and d^u.

The run times corresponding to these tests are summarized in Table 3. We were able to solve all of the problems in less than 75 seconds; Potts and Van Wassenhove could not solve all of the instances. The minimum, median, mean and maximum runtimes are very close to the same as the earlier sets of 1000-job problems. We again note that no reduction algorithm was used to limit the number of nodes and subsequently the branching. Furthermore, the heuristic, which is extremely fast, found the optimal for 144 of the 280 instances. As seen from Table 4, the quality of the heuristic was similar to that of the earlier 1000-job problem sets.

5. Summary

This paper contains results on scheduling a single machine in order to minimize the weighted number of tardy jobs. A heuristic and an exact algorithm are proposed. Extensive experimentation indicates that the heuristic provides optimal solutions in most instances. It provides good solutions to problems with 2500 jobs in a few seconds. In addition, the computational testing has shown that the exact method solves typical problems with 2500 jobs optimally in about 6 minutes and the hardest instances in about 12 minutes on a desktop computer. Furthermore, these results provide insight into how the problem parameters influence solution difficulty and indicate the algorithm is robust with respect to difficult problems.

Acknowledgements

We would like to thank the referees for their helpful comments on this paper. Parts of this work were supported by grants from the NSF International Collaborative Research Division and from the NSF Division of Design, Manufacture & Industrial Innovation.

References

Dauzere Peres, S., 1995. Minimizing late jobs in the general one machine scheduling problem. European Journal of Operational Research 81, 134–142.

Jackson, J.R., 1955. Scheduling a production line to minimize maximum tardiness, Research Report 43, Management Science Research Project, University of California, Los Angeles, CA.

Karp, R.M., 1972. Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (Eds.), Complexity of Computer Computations. Plenum Press, New York, pp. 85–103.

Lawler, E.M., Moore, J.M., 1969. A functional equation and its application to resource allocation and sequencing problems. Management Science 16, 77–84.

Moore, J.M., 1968. An *n* job, one machine sequencing algorithm for minimizing the number of late jobs. Management Science 15, 102–109.

Parker, R.G., Rardin, R.L., 1988. Discrete Optimization. Academic Press, San Diego, CA.

Potts, C.N., Wassenhove, L.N., 1988. Algorithms for scheduling a single machine to minimize the weighted number of late jobs. Management Science 34, 843–858.

Sahni, S., 1976. Algorithms for scheduling independent jobs. Journal of the Association of Computing Machinery 23, 116–127.

Tang, G., 1990. A new branch and bound algorithm for minimizing the weighted number of tardy jobs. Annals of Operations Research 24, 225–232.

Villarreal, F.J., Bulfin, R.L., 1983. Scheduling a single machine to minimize the weighted number of tardy jobs. IIE Transactions 15 (4), 337–343.

Woolsey, R.E.D., Swanson, H.S., 1975. Operations Research for Immediate Application: A Quick and Dirty Manual. Harper and Row Publishers, New York.