We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

5

Minimizing the Weighted Number of Late Jobs with Batch Setup Times and Delivery Costs on a Single Machine

George Steiner and Rui Zhang[1]
DeGroote School of Business, McMaster University
Canada

1. Introduction

We study a single machine scheduling problem with batch setup time and batch delivery cost. In this problem, n jobs have to be scheduled on a single machine and delivered to a customer. Each job has a due date, a processing time and a weight. To save delivery cost, several jobs can be delivered together as a batch including the late jobs. The completion (delivery) time of each job in the same batch coincides with the batch completion (delivery) time. A batch setup time has to be added before processing the first job in each batch. The objective is to find a batching schedule which minimizes the sum of the weighted number of late jobs and the delivery cost. Since the problem of minimizing the weighted number of late jobs on a single machine is already $\mathcal{NP}$-hard [Karp, 1972], the above problem is also $\mathcal{NP}$-hard. We propose a new dynamic programming algorithm *(DP),* which runs in pseudopolynomial time. The *DP* runs in $O(n^5)$ time for the special cases of equal processing times or equal weights. By combining the techniques of binary range search and static interval partitioning, we convert the *DP* into a fully polynomial time approximation scheme *(FPTAS)* for the general case. The time complexity of this *FPTAS* is $O(n^4/\varepsilon + n^4 \log n)$.

Minimizing the total weighted number of late jobs on a single machine, denoted by $1||\sum w_j U_j$ [Graham *et. al,* 1979], is a classic scheduling problem that has been well studied in the last forty years. Moore [1968] proposed an algorithm for solving the unweighted problem on n jobs in $O(n \log n)$ time. The weighted problem was in the original list of $\mathcal{NP}$-hard problems of Karp [1972]. Sahni [1976] presented a dynamic program and a fully polynomial time approximation scheme (*FPTAS*) for the maximization version of the weighted problem in which we want to maximize the total weight of on-time jobs. Gens and Levner [1979] developed an *FPTAS* solving the minimization version of the weighted problem in $O(n^3/\varepsilon)$ time. Later on, they developed another *FPTAS* that improved the time complexity to $O(n^2 \log n + n^2/\varepsilon)$ [Gens and Levner, 1981].

In the batching version of the problem, denoted by $1|s|\sum w_j U_j$, jobs are processed in batches which require setup time s, and every job's completion time is the completion time of the last job in its batch. Hochbaum and Landy [1994] proposed a dynamic programming algorithm for this problem, which runs in pseudopolynomial time. Brucker and Kovalyov

[1] email:steiner@mcmaster.ca, zhangr6@mcmaster.ca

Source: Multiprocessor Scheduling: Theory and Applications, Book edited by Eugene Levner,
ISBN 978-3-902613-02-8, pp.436, December 2007, Itech Education and Publishing, Vienna, Austria

[1996] presented another dynamic programming algorithm for the same problem, which was then converted into an *FPTAS* with complexity $O(n^3/\varepsilon + n^3 \log n)$.

In this paper, we study the batch delivery version of the problem in which each job must be delivered to the customer in batches and incurs a delivery cost. Extending the classical three-field notation [Graham *et. al.,* 1979], this problem can be denoted by $1|s,q|\sum w_j U_j + bq$, where b is the total number of batches and q is the batch delivery cost. The model, without the batch setup times, is similar to the single-customer version of the supplier's supply chain scheduling problem introduced by Hall and Potts [2003] in which the scheduling component of the objective is the minimization of the sum of the weighted number of late jobs (late job penalties). They show that the problem is $\mathcal{NP}$-hard in the ordinary sense by presenting pseudopolynomial dynamic programming algorithms for both the single-and multi-customer case [Hall and Potts, 2003]. For the case of identical weights, the algorithms become polynomial. However, citing technical difficulties in scheduling late jobs for delivery [Hall and Potts, 2003] and [Hall, 2006], they gave pseudopolynomial solutions for the version of the problem where only *early* jobs get delivered. The version of the problem in which the late jobs also have to be delivered is more complex, as late jobs may need to be delivered together with some early jobs in order to minimize the batch delivery costs. In Hall and Potts [2005], the simplifying assumption was made that late jobs are delivered in a separate batch at the end of the schedule. Steiner and Zhang [2007] presented a pseudopolynomial dynamic programming solution for the multi-customer version of the problem which included the unrestricted delivery of late jobs. This proved that the problem with late deliveries is also $\mathcal{NP}$-hard only in the ordinary sense. However, the algorithm had the undesirable property of having the (fixed) number of customers in the exponent of its complexity function. Furthermore, it does not seem to be convertible into an *FPTAS.* In this paper, we present for $1|s,q|\sum w_j U_j + bq$ a different dynamic programming algorithm with improved pseudopolynomial complexity that also schedules the late jobs for delivery. Furthermore, the algorithm runs in polynomial time in the special cases of equal tardiness costs or equal processing times for the jobs. This proves that the polynomial solvability of $1||\sum U_j$ can be extended to $1|s,q|\sum U_j + bq$, albeit by a completely different algorithm. We also show that the new algorithm for the general case can be converted into an *FPTAS.*

The paper is organized as follows. In section 2, we define the $1|s,q|\sum w_j U_j + bq$ problem in detail and discuss the structure of optimal schedules. In section 3, we propose our new dynamic programming algorithm for the problem, which runs in pseudopolynomial time. We also show that the algorithm becomes polynomial for the special cases when jobs have equal weights or equal processing times. In the next section, we develop a three-step fully polynomial time approximation scheme, which runs in $O(n^4/\varepsilon + n^4 \log n)$ time. The last section contains our concluding remarks.

2. Problem definition and preliminaries

The problem can be defined in detail as follows. We are given n jobs, $J = \{1,2,..., n\}$, with processing time p_j, weight w_j, delivery due date $\bar{d}_j \geq p_j$, $j \in J$. Jobs have to be scheduled nonpreemptively on a single machine and delivered to the customer in batches. Several jobs could be scheduled and delivered together as a batch with a batch delivery cost q and delivery time τ. For each batch, a batch setup time s has to be added before processing the first job of the batch. Our goal is to find a batching schedule that minimizes the sum of the

weighted number of late jobs and delivery costs. Without loss of generality, we assume that all data are nonnegative integers.

A job is late if it is delivered after its delivery due date, otherwise it is early. The *batch completion time* is defined as the completion time of the last job in the batch on the machine. Since the delivery of batches can happen simultaneously with the processing of some other jobs on the machine, it is easy to see that a job is late if and only if its batch completion time is greater than its delivery due date minus τ. This means that each job j has an *implied due date* $d_j = \bar{d}_j - \tau$ on the machine. This implies that we do not need to explicitly schedule the delivery times and consider the delivery due dates, we can just use the implied due dates, or due dates in short, and job j is late if its batch completion time is greater than d_j. (From this point on, we use the term due date always for the d_j.) A batch is called an *early batch* if all jobs are early in this batch, it is called a *late batch* if every job is late in this batch, and a batch is referred to as *mixed batch* if it contains both early and late jobs. The *batch due date* is defined as the smallest due date of any job in the batch. The following simple observations characterize the structure of optimal schedules we will search for. They represent adaptations of known properties for the version of the problem in which there are no delivery costs and/or late jobs do not need to be delivered.

Proposition 2.1. *There exists an optimal schedule in which all early jobs are ordered in EDD (earliest due date first) order within each batch.*

Proof. Since all jobs in the same batch have the same batch completion time and batch due date, the sequencing of jobs within a batch is immaterial and can be assumed to be *EDD*.

Proposition 2.2. *There exists an optimal schedule in which all late jobs (if any) are scheduled in the last batch (either in a late batch or in a mixed batch that includes early jobs).*

Proof. Suppose that there is a late job in a batch which is scheduled before the last batch in an optimal schedule. If we move this job into this last batch, it will not increase the cost of the schedule.

Proposition 2.3. *There exists an optimal schedule in which all early batches are scheduled in EDD order with respect to their batch due date.*

Proof. Suppose that there are two early batches in an optimal schedule with batch completion times $t_i < t_k$ and batch due dates $d_i > d_k$. Since all jobs in both batches are early, we have $d_i > d_k \geq t_k > t_i$. Thus if we schedule batch k before batch i, it does not increase the cost of the schedule.

Proposition 2.4. *There exists an optimal schedule such that if the last batch of the schedule is not a late batch, i.e., there is at least one early job in it, then all jobs whose due dates are greater than or equal to the batch completion time are scheduled in this last batch as early jobs.*

Proof. Let the batch completion time of the last batch be t. Since the last batch is not a late batch, there must be at least one early job in this last batch whose due date is greater than or equal to t. If there is another job whose due date is greater than or equal to t but it was scheduled in an earlier batch, then we can simply move this job into this last batch without increasing the cost of the schedule.

Proposition 2.2 implies that the jobs which are first scheduled as late jobs can always be scheduled in the last batch when completing a partial schedule that contains only early jobs. The dynamic programming algorithm we present below uses this fact by generating all possible schedules on early jobs only and *designating* and putting aside the late jobs, which get scheduled only at the end in the last batch. It is important to note that when a job is designated to be late in a partial schedule, then its weighted tardiness penalty is added to the cost of the partial schedule.

3. The dynamic programming algorithm

The known dynamic programming algorithms for $1|s|\sum w_jU_j$ do not have a straightforward extension to $1|s,q|\sum w_jU_j + bq$, because the delivery of late jobs complicates the matter. We know that late jobs can be delivered in the last batch, but setting them up in a separate batch could add the potentially unnecessary delivery cost q for this batch when in certain schedules it may be possible to deliver late jobs together with early jobs and save their delivery cost. Our dynamic programming algorithm gets around this problem by using the concept of designated late jobs, whose batch assignment will be determined only at the end. Without loss of generality, assume that the jobs are in *EDD* order, i.e., $d_1 \le d_2 \le \ldots \le d_n$ and let $P = \sum_{j=1}^{n} p_j$. If $d_1 \ge P + s$, then it is easy to see that scheduling all jobs in a single batch will result in no late job, and this will be an optimal schedule. Therefore, we exclude this trivial case by assuming for the remainder of the paper that some jobs are due before $P + s$. The state space used to represent a partial schedule in our dynamic programming algorithm is described by five entries {*k, b, t, d, v*}:

k: the partial schedule is on the job set {1,2,..., *k*}, and it schedules some of these jobs as early while only *designating* the rest as late;

b: the number of batches in the partial schedule;

t: the batch completion time of the last scheduled batch in the partial schedule;

d: the due date of the last batch in the partial schedule;

v: the cost (value) of the partial schedule.

Before we describe the dynamic programming algorithm in detail, let us consider how we can reduce the state space. Consider any two states (k, b, t_1, d, v_1) and (k, b, t_2, d, v_2). Without loss of generality, let $t_1 \le t_2$. If $v_1 \le v_2$, we can eliminate the second state because any later states which could be generated from the second state can not lead to better v value than the value of similar states generated from the first state. This validates the following elimination rule, and a similar argument could be used to justify the second remark.

Remark 3.1. For any two states with the same entries {*k,b,t,d,* }, we can eliminate the state with larger *v*.

Remark 3.2. For any two states with the same entries {*k, b, ,d,v*}, we can eliminate the state with larger *t*.

The algorithm recursively generates the states for the partial schedules on batches of *early* jobs and at the same time designates some other jobs to be late without actually scheduling these late jobs. The jobs designated late will be added in the last batch at the time when the partial schedule gets completed into a full schedule. The tardiness penalty for every job designated late gets added to the state variable *v at the time* of designation. We look for an optimal schedule that satisfies the properties described in the propositions of the previous section. By Proposition 2.2, the late jobs should all be in the last batch of a full schedule. It is equivalent to say that any partial schedule *{k, b, t, d, v}* with $1 \le b \le n - 1$ can be completed into a full schedule by one of the following two ways:

1. Add all unscheduled jobs {$k + 1, k + 2,..., n$} and the previously designated late jobs to the end of the last batch *b if* the resulting batch completion time *(P + bs)* does not exceed the batch due date *d* (we call this a *simple completion);* or
2. Open a new batch b+1, and add all unscheduled jobs {$k + 1, k + 2,..., n$} and the previously designated late jobs to the schedule in this batch. (We will call this a *direct completion.)*

We have to be careful, however, as putting a previously designated late job into the last batch this way may make such a job actually early if its completion time (P+*bs* or $P + (b + 1)$ *s*, respectively) is not greater than its due date. This situation would require rescheduling such a designated late job among the early jobs and removing its tardiness penalty from the cost *v*. Unfortunately, such rescheduling is not possible, since we do not know the identity of the designated late jobs from the state variables (we could only derive their total length and tardy weight). The main insight behind our approach is that there are certain *special states,* that we will characterize, whose completion never requires such a rescheduling. We proceed with the definition of these special states.

It is clear that a full schedule containing exactly l ($1 \le l \le n$) batches will have its last batch completed at $P + ls$. We consider all these possible completion times and define certain *marker jobs* m_i and *batch counters* g_i in the *EDD* sequence as follows: Let m_0 be the last job with $d_{m_0} < P + s$ and m_0 +1 the first job with $d_{m_0+1} \ge P+s$. If m_0 +1 does not exist, i.e., $m_0 = n$, then we do not need to define any other marker jobs, all due dates are less than $P + s$, and we will discuss this case separately later. Otherwise, define $g_0 = 0$ and let $g_1 \ge 1$ be the largest integer for which $d_{m_0+1} \ge P + g_1 s$. Let the marker job associated with g_1 be the job $m_1 \ge m_0 + 1$ whose due date is the largest due date strictly less than $P + (g_1 + 1)s$, i.e., $d_{m_1} < P + (g_1 + 1)s$ and $d_{m_1+1} \ge P + (g_1 + 1)s$. Define recursively for $i = 2,3,\ldots,h-1$, $g_i \ge g_{i-1} + 1$ to be the smallest counter for which there is a marker job $m_i \ge m_{i-1} + 1$ such that $d_{m_i} < P + (g_i + 1)\, s$ and $d_{m_i+1} \ge P + (g_i + 1)\, s$. The last marker job is $m_h = n$ and its counter g_h is the largest integer for which $P + g_h s \le d_n < P + (g_h + 1)s$. We also define $g_{h+1} = g_h + 1$. Since the maximum completion time to be considered is $P+ns$ for all possible schedules (when every job forms a separate batch), any due dates which are greater than or equal to $P + ns$ can be reduced to $P + ns$ without affecting the solution. Thus we assume that $d_n \le P+ns$ for the rest of the paper, which also implies $g_h + 1 \le n+1$.

For convenience, let us also define $T_{1,0} = P + g_1 s$, $T_{i,k} = P + (g_i + k)s$ for $i = 1,\ldots, h$ and $k = 0,1,\ldots, k(i)$, where each $k(i)$ is the number for which $T_{i,\,k\,(i)} = P + (g_i + k(i))s = P + g_{i+1}\, s = T_{i+1,0}$, and $T_{h,1} = P + (g_h + 1)s$. Note that this partitions the time horizon $[P,\ P + (g_h + 1)s]$ into consecutive intervals of length *s*. We demonstrate these definitions in Figure 1.

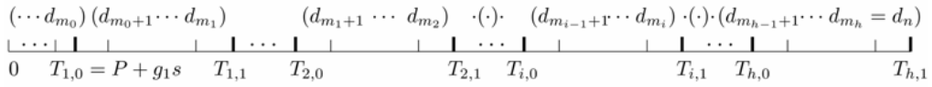


Figure 1. Marker Jobs and Corresponding Intervals

We can distinguish the following two cases for these intervals:

1. $T_{i,1} = T_{i+1,0}$, i.e., $k(i) = 1$: This means that the interval immediately following $I_i = [T_{i,0},\ T_{i,1})$ contains a due date. This implies that $g_{i+1} = g_i + 1$;
2. $T_{i,1} \ne T_{i+1,0}$, i.e., $k(i) > 1$: This means that there are $k(i) - 1$ intervals of length s starting at $P + (g_i + 1)s$ in which no job due date is located.

In either case, it follows that every job $j > m_0$ has its due date in one of the intervals $I_i = [T_{i,0},\ T_{i,1})$ for some $i \in \{1,\ldots, h\}$, and the intervals $[T_{i,l},\ T_{i,l+1})$ contain no due date for $i = 1,\ldots,h$ and $l>0$.

Figure 1 shows that jobs from m_0+1 to m_1 have their due date in the interval $[T_{1,0},\ T_{1,1})$. Each marker job m_i is the last job that has its due date in the interval $I_i = [T_{i,0},\ T_{i,1})$ for $i = 1,\ldots, h$, i.e., we have $T_{i,0} \le d_{m_{i-1}+1} \le d_{m_{i-1}+2} \le \ldots \le d_{m_i} < T_{i,1}$.

Now let us group all jobs into h +1 non-overlapping job sets $G_0 = \{1,..., m_0\}$, $G_1 = \{m_0 + 1,..., m_1\}$ and $G_i = \{m_{i-1} + 1,..., m_i\}$ for $i = 2,..., h$. Then we have $d_j \in I_i \ \forall j \in G_i$ and $i \geq 1$. We also define the job sets $J_0 = Go$, $J_i = G_0 \cup G_1 \cup ... \cup G_i$, for $i = 1,2,..., h-1$ and $J_h = G_0 \cup G_1 \cup ... \cup G_h = J$.
The *special states* for *DP* are defined by the fact that their (k, b) state variables belong to the set H defined below:
If $m_0 = n$, then let $H = \{(n, 1), (n, 2), ..., (n, n-1)\}$;
If $m_0 < n$, then let $H = H_1 \cup H_2 \cup H_3$, where

1. If $g_1 > 1$, then $H_1 = \{(m_0, 1), (m_0, 2), ..., (m_0, g_1-1)\}$, otherwise $H_1 = \emptyset$;
2. $H_2 = \{(m_1, g_1), (m_1, g_1+1), ..., (m_1, g_2-1), (m_2, g_2), (m_2, g_2+1), ..., (m_2, g_3-1), ..., (m_i, g_i), (m_i, g_i+1), ..., (m_i, g_{i+1}-1), ..., (m_{h-1}, g_{h-1}), ..., (m_{h-1}, g_h-1)\}$;
3. If $1 < g_h < n$, then $H_3 = \{(n, g_h), (n, g_h+1), ..., (n, n-1)\}$, otherwise $H_3 = \emptyset$.

Note that $m_h = n$ and thus the pairs in H_3 follow the same pattern as the pairs in the other parts of H. The dynamic program follows the general framework originally presented by Sahni [1976].

The Dynamic Programming Algorithm *DP*

[Initialization] Start with jobs in *EDD* order

1. Set $(0, 0, 0, 0, 0) \in S^{(0)}$, $S^{(k)} = \emptyset$, $k = 1, 2, ..., n$, $T^* = \emptyset$, and define m_0, g_i and m_i, $i = 1,2,..., h$;
2. If $m_0 + 1$ does not exist, i.e., $m_0 = n$, then set $H = \{(n, 1), (n, 2), ..., (n, n-1)\}$; Otherwise set $H = H_1 \cup H_2 \cup H_3$.

Let $I = \{(k, b) | 1 \leq b \leq k \leq n\}$ the set of all possible pairs and $\bar{H} = I - H$, the complementary set of H.

[Generation] Generate set $S^{(k)}$ **for** $k = 1$ to $n + 1$ from $S^{(k-1)}$ as follows:

Set $T = \emptyset$;

[Operations] **Do** the following **for** each state $(k-1, b, t, d, v)$ in $S^{(k-1)}$

Case $(k - 1, b) \in H$

1. *If $t < P + bs$, set $T^* = T^* \cup (n, b+1, P + (b+1)s, d', v + q)$ /* Generate the direct completion schedule and add it to the solution set T^*, where d' is defined as the due date of the first job in batch $b+1$;*
2. *If $t = P + bs$, set $T^* = T^* \cup (n, b, P + bs, d, v)$ /* We have a partial schedule in which all jobs are early. (This can happen only when $k - 1 = n$.)*

Case $(k - 1, b) \in \bar{H}$

1. *If $t + p_k \leq d$ and $k \leq n$, set $T = T \cup (k, b, t + p_k, d, v)$ /* Schedule job k as an early job in the current batch;*
2. *If $t + p_k + s \leq d_k$ and $k \leq n$, set $T = T \cup (k, b+1, t + p_k + s, d_k, v + q)$ /* Schedule job k as an early job in a new batch;*
3. *If $k \leq n$, set $T = T \cup (k, b, t, d, v + w_k)$ /* Designate job k as a late job by adding its weight to v and reconsider it at the end in direct completions.*

Endfor

[Elimination] Update set $S^{(k)}$

1. For any two states (k, b, t, d, v) and (k, b, t, d, v') with $v \leq v'$, eliminate the one with v' from set T based on Remark 3.1;
2. For any two states (k, b, t, d, v) and (k, b, t', d, v) with $t \leq t'$, eliminate the one with t' from set T based on Remark 3.2;
3. Set $S^{(k)} = T$.

Endfor

[*Result*] The optimal solution is the state with the smallest v in the set T^*. Find the optimal schedule by backtracking through all ancestors of this state.

We prove the correctness of the algorithm by a series of lemmas, which establish the crucial properties for the special states.

Lemma 3.1. *Consider a partial schedule (m_i, b, t, d, v) on job set J_i, where $(m_i, b) \in H$. If its completion into a full schedule has b+1 batches, then the final cost of this completion is exactly $v + q$.*

Proof. We note that completing a partial schedule on b batches into a full schedule on $b + 1$ batches means a direct completion, i.e., all the unscheduled jobs (the jobs in $J - J_i$, if any) and all the previously designated late jobs (if any) are put into batch b+1, with completion time $P + (b + 1)s$.

Since all the previously designated late jobs are from J_i for a partial schedule (m_i, b, t, d, v), their due dates are not greater than $d_{m_i} < P + (g_i + 1)s \leq P + (b + 1)s$. Therefore, all designated late jobs stay late when scheduled in batch b+1. Next we show that unscheduled jobs $j \in (J - J_i)$ must be early in batch b+1. We have three cases to consider.

Case 1. $m_0 = n$ *and* $i = 0$:

In this case, $H = \{(n, 1), (n, 2), \ldots, (n, n - 1)\}$ and $J_0 = J$, i.e. all jobs have been scheduled early or designated late in the state (m_0, b, t, d, v). Therefore, there are no unscheduled jobs.

Case 2. $m_0 < n$ *and* $b = g_i$:

Since $g_0 = 0$ by definition, we must have $i \geq 1$ in this case. The first unscheduled job $j \in (J - J_i)$ is job $m_i + 1$ with due date $d_{m_i+1} \geq P + (g_i + 1)s = P + (b + 1)s$. Thus $m_i + 1$ and all other jobs from $J - J_i$ have a due date that is at least $P + (b + 1)s$, and therefore they will all be early in batch b+1.

Case 3. $m_0 < n$ *and* $b > g_i$:

This case is just an extension of the case of $b = g_i$.

If $i = 0$, then the first unscheduled job for the state (m_0, b, t, d, v) is $m_0 + 1$. Thus every unscheduled job j has a due date $d_j \geq d_{m_0+1} \geq P + g_1 s \geq P + (b + 1)s$, where the last inequality holds since $(m_0, b) \in H_i$ and therefore, $b \leq g_1 - 1$.

If $1 \leq i < h$, then we cannot have $k(i) = 1$: By definition, if $k(i) = 1$, then $g_i + k(i) - 1 = g_i = g_{i+1} - 1$, which contradicts $b > g_i$ and $(m_i, b) \in H$. Therefore, we must have $k(i) > 1$, and b could be any value from $\{g_i + 1, \ldots, g_i + k(i) - 1\}$. This means that $P + (b + 1)s < P + (g_i + k(i))s = P + g_{i+1} s$. We know, however, that every unscheduled job has a due date that is at least $T_{i+1, 0} = P + g_{i+1} s$. Thus every job from $J - J_i$ will be early indeed.

If $i = h$, then we have $m_h = n$ and $J_h = J$, and thus all jobs have been scheduled early or designated late in the state (m_i, b, t, d, v). Therefore, there are no unscheduled jobs.

In summary, we have proved that all previously designated late jobs (if any) remain late in batch b+1, and all jobs from $J - J_i$ (if any) will be early. This means that v correctly accounts for the lateness cost of the completed schedule, and we need to add to it only the delivery cost q for the additional batch b+1. Thus the cost of the completed schedule is $v + q$ indeed.

Lemma 3.2. *Consider a partial schedule (m_i, b, t, d, v) on job set J_i, where $(m_i, b) \in H$ and $b \neq n - 1$. Then any completion into a full schedule with more than $b + 1$ batches has a cost that is at least $v + q$, i.e., the direct completion has the minimum cost among all such completions of (m_i, b, t, d, v).*

Proof. If $m_i = n$, then the partial schedule is of the form (n, b, t, d, v), $(n, b) \in H$, $b \neq n - 1$. (This implies that either $m_0 = n$ with $i = 0$ or $(m_i, b) \in H_3$ with $i = h$.) Since there is no unscheduled job left, all the new batches in any completion are for previously designated late jobs. And since all the previously designated late jobs have due dates that are not greater than

$d_n < P + (g_i + 1)s \leq P + (b+1)s$, these jobs will stay late in the completion. The number of new batches makes no difference to the tardiness penalty cost of late jobs. Therefore, the best strategy is to open only one batch with cost *q*. Thus the final cost of the direct completion is minimum with cost $v + q$.

Consider now a partial schedule (m_i, *b, t, d, v*), (m_i, *b*) $\in H$, $b \neq n-1$ when $m_i < n$. Since all the previously designated late jobs (if any) are from J_i, their due dates are not greater than $d_{m_i} \leq P + g_{i+1}s \leq P + (b+1)s$. Furthermore, since all unscheduled jobs are from $J - J_i$, their due dates are not less than $d_{m_i+1} \geq P + g_{i+1}s \geq P + (b+1)s$. Thus scheduling all of these jobs into batch $b + 1$ makes them early without increasing the tardiness cost. It is clear that this is the best we can do for completing (m_i, *b, t, d, v*) into a schedule with $b + 1$ or more batches. Thus the final cost of the direct completion is minimum again with cost $v + q$.

Lemma 3.3. *Consider a partial schedule (m_i, b, t, d, v} on job set J_i ($i \geq 1$), where (m_i, b) $\in H$ and b > 1. If it has a completion into a full schedule with exactly b batches and cost v', then there must exist either a partial schedule $(m_i, b-1, \bar{t}, \bar{d}, \bar{v})$ whose direct completion is of the same cost v' or there exists a partial schedule $(m_{i-1}, b-1, \bar{t}, \bar{d}, \bar{v})$ whose direct completion is of the same cost v'.*

Proof. To complete the partial schedule (m_i,*b,t,d,v*) into a full schedule on *b* batches, all designated late jobs and unscheduled jobs have to be added into batch *b.*

Case 1. $b > g_i$:

Let us denote the early jobs by $E_i \subset J_i$ *in* batch *b* in the partial schedule (m_i, *b, t, d, v*). Adding the designated late jobs and unscheduled jobs to batch *b* will result in a batch completion time of *P+bs.* This makes all jobs in E_i late since $d_j \leq d_{m_i} < P+(g_i+1)s \leq P+bs$ for $j \in E_i$. Thus the cost of the full schedule should be $v' = v + \sum_{j \in E_i} w_j$. We cannot do this calculation, however, since there is no information available in *DP* about what E_i is. But if we consider the partial schedule $(m_i, b-1, \bar{t}, \bar{d}, \bar{v})$ = $(m_i, b-1, t - \sum_{j \in E_i} p_j, d, v + \sum_{j \in E_i} w_j - q)$ with one less batch, where $\bar{d}$ is the smallest due date in batch $b - 1$ in the partial schedule (m_i, *b, t, d, v),* the final cost of the *direct* completion of the partial schedule $(m_i, b-1, t - \sum_{j \in E_i} p_j, \bar{d}, v + \sum_{j \in E_i} w_j - q)$ would be exactly $v' = v + \sum_{j \in E_i} w_j$ by Lemma 3.1. We show next that this partial schedule $(m_i, b-1, t - \sum_{j \in E_i} p_j, \bar{d}, v + \sum_{j \in E_i} w_j - q)$ does get generated in the algorithm.

In order to see that *DP* will generate the partial schedule $(m_i, b-1, t - \sum_{j \in E_i} p_j, \bar{d}, v + \sum_{j \in E_i} w_j - q)$ suppose that during the generation of the partial schedule (m_i, *b, t, d, v*), *DP* starts batch *b* by adding a job *k* as early. This implies that the jobs that *DP* designates as late on the path of states leading to (m_i, *b, t, d, v)* are in the set $L_i = \{k, k+1, ..., m_i\} - E_i$. In other words, *DP* has in the path of generation for (m_i,*b,t,d,v*) a partial schedule $(k-1, b-1, t - \sum_{j \in E_i} p_j, \bar{d}, v - \sum_{j \in L_i} w_j - q)$. Then it will also generate from $(k-1, b-1, t - \sum_{j \in E_i} p_j, \bar{d}, v - \sum_{j \in L_i} w_j - q)$ the partial schedule $(m_i, b-1, t - \sum_{j \in E_i} p_j, \bar{d}, v + \sum_{j \in E_i} w_j - q)$ by simply designating *all* jobs in $E_i \cup L_i$ as late.

Case 2. $b = g_i \neq 1$:

Suppose the partial schedule (m_i, *b, t, d, v*) has in batch *b* the sets of early jobs $E_{i-1} \cup E$, where $E_{i-1} \subset J_{i-1}$ and $E \subset (J_i - J_{i-1})$. Adding the designated late jobs and unscheduled jobs to batch *b* will result in a batch completion time of *P + bs.* This makes all jobs in E_{i-1} late since $d_j \leq d_{m_{i-1}} < P + g_i s$ for $j \in E_{i-1}$. On the other hand, if $L \subset (J_i - J_{i-1} - E)$ denotes the previously designated late jobs from $J_i - J_{i-1}$ in (m_i, *b, t, d, v*), then these jobs become early since $P + g_i s \leq d_{m_{i-1}+1} \leq d_j$ for $j \in L$. For similar reasons, all previously designated late jobs *not* in *L* stay late, jobs in *E* remain early and all other jobs from $J - J_i$ will be early too. In summary, the cost for the full completed schedule derived from (m_i,*b,t,d,v*) should be $v' = v + \sum_{j \in E_{i-1}} w_j - \sum_{j \in L} w_j$. Again, we cannot do this calculation, since

there is no information about E_{i-1} and L. However, suppose that $E_{i-1} \neq \emptyset$, and consider the partial schedule $(m_{i-1}, b-1, \bar{t}, \bar{d}, \bar{v})$ = $(m_{i-1}, b-1, t-\sum_{j\in E\cup E_{i-1}} p_j, \bar{d}, v+ \sum_{j\in E_{i-1}} w_j - \sum_{j\in L} w_j - q)$ with one less batch, where d is the smallest due date in batch $b-1$ in the partial schedule (m_i, b, t, d, v). The final cost of the *direct* completion of the partial schedule $(m_{i-1}, b-1, t-\sum_{j\in E\cup E_{i-1}} p_j, \bar{d}, v+\sum_{j\in E_{i-1}} w_j - \sum_{j\in L} w_j - q)$ would be exactly $v' = v + \sum_{j\in E_{i-1}} w_j - \sum_{j\in L} w_j$ by Lemma 3.1. Next, we show that this partial schedule $(m_{i-1}, b-1, t-\sum_{j\in E\cup E_{i-1}} p_j, \bar{d}, v+\sum_{j\in E_{i-1}} w_j - \sum_{j\in L} w_j - q)$ does get generated during the execution of *DP*.

To see the existence of the partial schedule $(m_{i-1}, b-1, \bar{t}, \bar{d}, \bar{v})$ = $(m_{i-1}, b-1, t-\sum_{j\in E\cup E_{i-1}} p_j, \bar{d}, v+\sum_{j\in E_{i-1}} w_j - \sum_{j\in L} w_j - q)$ note that *DP* must start batch b on the path of states leading to (m_i, b, t, d, v) by scheduling a job $k \leq m_{i-1}$ early in iteration k from a state $(k-1, b-1, t-\sum_{j\in E_i\cup E} p_j, \bar{d}, v-(\sum_{j=k}^{m_{i-1}} w_j - \sum_{j\in E_{i-1}} w_j) - \sum_{j\in L} w_j - q)$ (We cannot have $k > m_{i-1}$ since this would contradict $E_{i-1} \neq \emptyset$. Note also that $(\sum_{j=k}^{m_{i-1}} w_j - \sum_{j\in E_{i-1}} w_j)$ accounts for the weight of those jobs from {k, k+1,..., m_{i-1}} that got designated late between iterations k and m_{i-1} during the generation of the state (m_i,b,t,d,v).) In this case, it is clear that *DP* will also generate from $(k-1, b-1, t-\sum_{j\in E_i\cup E} p_j, \bar{d}, v- (\sum_{j=k}^{m_{i-1}} w_j - \sum_{j\in E_{i-1}} w_j) - \sum_{j\in L} w_j - q)$ a partial schedule on J_{i-1} in which *all* jobs in E_{i-1} are designated late, in addition to those jobs (if any) from {k, k+1,..., m_{i-1}} that are designated late in (m_i, b, t, d, v). Since this schedule will designate all of {k, k+1,..., m_{i-1}} late, the lateness cost of this set of jobs must be added, which results in a state $(m_{i-1}, b-1, t-\sum_{j\in E\cup E_{i-1}} p_j, \bar{d}, v+\sum_{j\in E_{i-1}} w_j - \sum_{j\in L} w_j - q)$. This is the state $(m_{i-1}, b-1, \bar{t}, \bar{d}, \bar{v})$ whose existence we claimed.

The remaining case is when $E_{i-1} = \emptyset$. In this case, batch b has no early jobs in the partial schedule (m_i,b,t,d,v) from the set J_{i-1} and if k again denotes the first early job in batch b, then $k \in J_i - J_{i-1}$. This clearly implies that (m_i,b,t,d,v) must have a parent partial schedule $(m_{i-1}, b-1, t-\sum_{j\in E} p_j, \bar{d}, v-\sum_{j\in L} w_j - q)$. Consider the direct completion of this schedule: All designated late jobs must come from J_{i-1} and thus they stay late with a completion time of $P + bs$. Furthermore, all jobs from $J - J_{i-1}$ will be early, and therefore, the cost of this direct completion will be $v - \sum_{j\in L} w_j - q = v'$.

The remaining special cases of $b = 1$, which are not covered by the preceding lemma, are (m_i, b) = (m_1, 1) or (m_i, b) = (m_0, 1), and they are easy: Since all jobs are delivered at the same time $P + s$, all jobs in J_0 or J, respectively, are late, and the rest of the jobs are early. Thus there is only one possible full schedule with cost $v' = \sum_{j=1}^{m_0} w_j + q$ or $v' = \sum_{j=1}^{n} w_j + q$.

In summary, consider any partial schedule (m_i, b, t, d, v) on job set J_i, where $(m_i, b) \in H$, or a partial schedule (n, b, t, d, v) on job set J and assume that the full schedule S' = (n, b' , $P + b's$, d' , v') is a completion of this partial schedule and has minimum cost v'. Then the following schedules generated by *DP* will contain a schedule among them with the same minimum cost as S':

1. the *direct completion* of (m_i,b,t,d,v), if (m_i, b) $\neq$ (m_i, q_i) and $b' > b$, by Lemma 3.1 and Lemma 3.2;
2. the *direct completion* of a partial schedule $(m_i, b-1, \bar{t}, \bar{d}, \bar{v})$, if ($m_i$, b) $\neq$ (m_i, q_i) and b' = b, by Lemma 3.3;
3. the *direct completion* of a partial schedule $(m_i, b-1, \bar{t}, \bar{d}, \bar{v})$, if ($m_i$, b) = (m_i, q_i), $i > 1$ and b' = b, by Lemma 3.3;
4. the full schedule $(n, 1, P+s, d_{m_0+1}, \sum_{j=1}^{m_0} w_j + q)$ if $m_0 < n$ and $b' \geq b = q_1 = 1$ i.e., (m_i, b) = (m_1, 1);

5. the full schedule $(n, 1, P + s, d_1, \sum_{j=1}^{n} w_j + q)$, if $m_0 = n$ and $b' \geq b = 1$. i.e., (m_i, b) = (m_0, 1).

Theorem 3.1. *The dynamic programming algorithm DP is a pseudopolynomial algorithm, which finds an optimal solution for* $1|s,q|\sum w_jU_j + bq$ *in* $O(n^3\min\{d_n, P + ns, W + nq\})$ *time and space, where* $P = \sum_{j=1}^{n} p_j$ *and* $W = \sum_{j=1}^{n} w_j$.

Proof. The correctness of the algorithm follows from the preceding lemmas and discussion.
It is clear that the time and space complexity of the procedures [*Initialization*] and [*Result*] is dominated by the [*Generation*] procedure. At the beginning of iteration *k,* the total number of possible values for the state variables {*k, b, t, d, v*} in $S^{(k)}$ is upperbounded as follows: *n* is the upper bound of *k* and *b; n* is the upper bound for the number of different *d* values; min{d_n, *P* + *ns*} is an upper bound of *t* and *W* + *nq* is an upper bound of *v*, and because of the elimination rules, min{d_n, *P*+*ns, W*+*nq*} is an upper bound for the number of different combinations of *t* and *v.* Thus the total number of different states at the beginning of each iteration *k* in the *[Generation]* procedure is at most $O(n^2 \min\{d_n, P + ns, W + nq\})$. In each iteration *k,* there are at most three new states generated from each state in $S^{(k-1)}$ and this takes constant time. Since there are *n* iterations, the [*Generations*] procedure could indeed be done in $O(n^3 \min\{d_n, P + ns, W + nq\})$ time and space.

Corollary 3.1. *For the case of equal weights, the dynamic programming algorithm DP finds an optimum solution in* $O(n^5)$ *time and space.*

Proof. For any state, *v* is the sum of two different cost components: the delivery costs from {*q*, 2*q*,..., *nq*} and the weighted number of late jobs from {0, *w*,..., *nw*}, where $w_j = w$, $\forall j \in J$. Therefore, *v* can take at most $n(n + 1)$ different values and the upper bound for the number of different states becomes $O(n^3 \min\{d_n, P + ns, n^2\}) = O(n^5)$.

Corollary 3.2. *For the case of equal processing times, the dynamic programming algorithm DP finds an optimum solution in* $O(n^5)$ *time and space.*

Proof. For any state, *t* is the sum of two different time components: the setup times from {*s*, ...,*ns*} and the processing times from {0,*p*, ...,*np*}, where $p_j = p$, $\forall j \in J$. Thus, *t* can take at most $n(n + 1)$ different values, and the upper bound for the number of different states becomes $O(n^3 \min\{d_n, n^2, W + nq\}) = O(n^5)$.

4. The Fully Polynomial Time Approximation Scheme

To develop a fully polynomial time approximation scheme *(FPTAS),* we will use static interval partitioning originally suggested by Sahni [1976] for maximization problems. The efficient implementation of this approach for minimization problems is more difficult, as it requires prior knowledge of a lower *(LB)* and upper bound *(UB)* for the unknown optimum value v^*, such that the *UB* is a constant multiple of *LB.* In order to develop such bounds, we propose first a range algorithm *R(u,*ε*),* which for given *u* and ε, either returns a full schedule with cost $v \leq u$ or verifies that $(1 - \varepsilon)\, u$ is a lower bound for the cost of any solution. In the second step, we use repeatedly the range algorithm in a binary search to narrow the range [*LB, UB*] so that $UB \leq 2LB$ at the end. Finally, we use static interval partitioning of the narrowed range in the algorithm *DP* to get the *FPTAS.* Similar techniques were used by Gens and Levner [1981] for the one-machine weighted-number-of-late-jobs problem $(1||\sum w_jU_j)$ and Brucker and Kovalyov [1996] for the one-machine weighted-number-of-late-jobs batching problem without delivery costs $(1|s, q = 0|\sum w_jU_j)$.
The range algorithm is very similar to the algorithm *DP* with a certain variation of the [*Elimination*] and [*Result*] procedures.

The Range Algorithm $R(u, \varepsilon)$
[*Initialization*] The same as that in the algorithm *DP*.
[*Partition*] Partition the interval [0, u] into $\lceil n/\varepsilon \rceil$ equal intervals of size $u\varepsilon/n$, with the last one possibly smaller.
[Generation] Generate set $S^{(k)}$ **for** k = 1 to $k = n + 1$ from $S^{(k-1)}$ as follows:
Set $\mathcal{T} = \emptyset$;
[*Operations*] The same as those in the algorithm *DP*.
[*Elimination*] Update set $S^{(k)}$
1. Eliminate any state (k, b, t, d, v) if $v > u$.
2. If more than one state has a v value that falls into the same interval, then discard all but one of these states, keeping only the *representative* state with the smallest t coordinate for each interval.
3. For any two states (k, b, t, d, v) and (k, b, t, d, v') with $v < v'$, eliminate the one with v' from set $\mathcal{T}$ based on Remark 3.2;
4. Set $S^{(k)} = \mathcal{T}$.

Endfor
[*Result*]
If $\mathcal{T}^* = \emptyset$, then $v^* > (1 - \varepsilon)\, u$;
If $\mathcal{T}^* \neq \emptyset$, then $v^* \leq u$.

Theorem 4.1. *If at the end of the range algorithm $R(u, \varepsilon)$, we found $\mathcal{T}^* = \emptyset$, then $v^* > (1-\varepsilon)u$; otherwise $v^* \leq u$. The algorithm runs in $O(n^4/\varepsilon)$ time and space.*

Proof. If $\mathcal{T}^*$ is not empty, then there is at least one state (n, b, t, d, v) that has not been eliminated. Therefore, v is in some subinterval of [0, u] and $v^* \leq v \leq u$. If $\mathcal{T}^* = \emptyset$, then all states with the first two entries $(k, b) \in H$ have been eliminated. Consider any feasible schedule (n,b,t,d,v). The fact that $\mathcal{T}^* = \emptyset$ means that any ancestor state of (n,b,t,d,v) with cost $\tilde{v} \leq v$ must have been eliminated at some iteration k in the algorithm either because $\tilde{v} > u$ or by interval partitioning, which kept some other representative state with cost $\tilde{v}'$ and maximum error $\varepsilon u/n$. In the first case, we also have $v > u$. In the second case,
let $v' \geq \tilde{v}'$ be the cost of a completion of the representative state and we must have $v' > u$ since $\mathcal{T}^* = \emptyset$. Since the error introduced in one iteration is at most $\varepsilon u/n$, the overall error is at most $n(\varepsilon u/n) = \varepsilon u$, i.e., $v \geq v' - n(\varepsilon u/n) = v' - \varepsilon u > u - \varepsilon u = (1 - \varepsilon)u$. Thus $v > (1 - \varepsilon)u$ for any feasible cost value v.
For the complexity, we note that $|\mathcal{S}^{(k)}| \leq \lceil n^3/\varepsilon \rceil$ for k = 1,2,...,n. Since all operations on a single state can be performed in $O(1)$ time, the overall time and space complexity is $O(n^4/\varepsilon)$.
The repeated application of the algorithm $R(u, \varepsilon)$ will allow us to narrow an initially wide range of upper and lower bounds to a range where our upper bound is only twice as large as the lower bound. We will start from an initial range $v' \leq v^* \leq nv'$. Next, we discuss how we can find such an initial lower bound v'.
Using the same data, we construct an auxiliary batch scheduling problem in which we want to minimize the maximum weight of late jobs, batches have the same batch-setup time s, the completion time of each job is the completion time of its batch, but there are no delivery costs. We denote this problem by $1|s, q = 0|\max w_j U_j$. It is clear that the minimum cost of this problem will be a lower bound for the optimal cost of our original problem.
To solve the $1|s, q = 0|\max w_j U_j$ problem, we first sort all jobs into smallest-weight-first order, i.e., $w_{[1]} \leq w_{[2]} \leq \ldots \leq w_{[n]}$. Here we are using [$k$] to denote the job with the kth smallest weight. Suppose that [k^*] has the largest weight among the late jobs in an optimal schedule. It is

clear that there is also an optimal schedule in which every job [*i*], for *i* = 1,2,..., *k**, is late, since we can always reschedule these jobs at the end of the optimal schedule without making its cost worse. It is also easy to see that we can assume without loss of generality that the early jobs are scheduled in *EDD* order in an optimal schedule. Thus we can restrict our search for an optimal schedule of the following form:

There is a $k \in \{0,1,..., n\}$ such that jobs {[*k* + 1],..., [*n*]} are early and they are scheduled in *EDD* order in the first part of the schedule, followed by jobs {[1], [2],..., [*k*]} in the last batch in any order. The existence of such a schedule can be checked by the following simple algorithm.

The Feasibility Checking Algorithm *FC*(*k*)

[*Initialization*] For the given *k* value, sort the jobs {[*k* + 1],..., [*n*]} into *EDD* order, and let this sequence be $(\theta_1, \theta_2, ..., \theta_f)$, where $f = n - k$.
Set $i = 1$, $j = \theta_1$, $t = s + p_j$ and $d = d_j$
If $t > d$, no feasible schedule exists and goto [*Report*];
If $t \le d$, set $i = 2$ and goto [*FeasibilityChecking*].

[*FeasibilityChecking*] **While** $i \le f$ **do**
Set $j = \theta_i$
If $t + p_j > d$, start a new batch for job *j*;
if $t + s + p_j > d_j$, no feasible schedule exists and goto [*Report*};
if $t+s+p_j \le d_j$, set $t = t+s+p_j$, $d = d_j$, $i = i+1$ and goto [*FeasibilityChecking*].
If $t + p_j \le$ d, set $t = t + p_j$, $i = i + 1$ and goto [*FeasibilityChecking*].
Endwhile

[*Report*] If $i \le f$, no feasible schedule exists. Otherwise, there exists a feasible batching schedule for jobs $(\theta_1, \theta_2, ..., \theta_f)$ in which these jobs are early.

The $1|s, q = 0|\max w_j U_j$ problem can be solved by repeatedly calling *FC*(*k*) for increasing *k* to find the first *k* value, denoted by *k**, for which *FC*(*k*) returns that a feasible schedule exists.

The Min-Max Weight Algorithm *MW*

[*Initialization*] Sort the jobs into a nondecreasing sequence by their weight $w_{[1]} \le w_{[2]} \le ... \le w_{[n]}$ and set $k = 0$.

[*AlgorithmFC*] **While** $k \le n$ call algorithm *FC*(*k*).
If *FC(k)* reports that no feasible schedule exists, set $k = k+1$ and goto [*AlgorithmFC*] ;
Otherwise, set $k^* = k$ and goto [*Result*];
Endwhile

[*Result*] If $k^* = 0$ then there is a schedule in which all jobs are early and set $w^* = 0$; otherwise, $w^* = w_{[k^*]}$ is the optimum.

Theorem 4.2. *The Min-Max Weight Algorithm MW finds the optimal solution to the problem* $1|s, q = 0|\max w_j U_j$ *in* $O(n^2)$ *time.*

Proof. For $k = 0$, *FC*(*k*) constructs the *EDD* sequence on the whole job set *J*, which requires $O(n\log n)$ time. We can obtain the sequence $(\theta_1, \theta_2, ..., \theta_{f-1})$ ($f = n - k$) in the initialization step of *FC*(*k* + 1), from the sequence $(\theta_1, \theta_2, ..., \theta_f)$ constructed for *FC*(*k*) in $O(n)$ time by simply deleting the job [*k*] from it. It is clear that all other operations in *FC(k)* need at most *O(n)* time. Since *MW* calls *FC*(*k*) at most (*n* + 1) times, the overall complexity of the algorithm is $O(n^2)$ indeed.

Corollary 4.1. *The optimal solution v* to the problem of minimizing the sum of the weighted number of late jobs and the batch delivery cost on a single machine,* $1|s, q|\sum_{j=1}^{n} w_j U_j + bq$, *is in the interval* [*v'*, *nv'*], *where v' = w* + q.*

Proof. It is easy to see that there is at least one batch and there are at most $n - k^* + 1$ batches in a feasible schedule. Also the weighted number of late jobs is at least w^* and at most $k^* w^*$

in an optimal schedule for $1|s,q|\sum_{j=1}^{n} w_j U_j + bq$. Thus $v' = w^* + q$ is a lower bound and $k^*w^* + (n - k^* + 1)q \le nw^* + nq = n(w^* + q) = nv'$ is an upper bound for the optimal solution v^* of $1|s,q|\sum_{j=1}^{n} w_j U_j + bq$.

Next, we show how to narrow the range of these bounds. Similarly to Gens and Levner [1981], we use the algorithm $R(u,\varepsilon)$ with $\varepsilon = 1/4$ in a binary search to narrow the range $[v', nv']$.

The Range and Bound Algorithm *RB*

[*Initialization*] Set $u' = nv'/2$;

[*BinarySearch*] Call $R(u', 1/4)$;

If $R(u', 1/4)$ reports that $v^* \le u'$, set $u' = u'/2$ and goto [*BinarySearch*];

If $R(u', 1/4)$ reports $v^* > 3u'/4$, set $u' = 3u'/2$.

[*Determination*] Call $R(u', 1/4)$.

If $R(u', 1/4)$ reports $v^* \le u'$, set $\overline{v} = u'/2$ and stop;

If $R(u', 1/4)$ reports $v^* > 3u'/4$, set $\overline{v} = 3u'/2$ and stop.

Theorem 4.3. T*he algorithm RB determines a lower bound* $\overline{v}$ *for* v^* *such that* $\overline{v} \le v^* \le 2\overline{v}$ *and it requires* $O(n^4 \log n)$ *time.*

Proof. It can be easily checked that when the algorithm stops, we have $\overline{v} \le v^* \le 2\overline{v}$. For each iteration of the range algorithm $R(u', 1/4)$, the whole value interval is divided into subintervals with equal length $\frac{u'}{4n}$ (the last subinterval may be less), where $u' \ge v'$. Since only values $v \le u'$ are considered in this range algorithm, the maximum number of subintervals is less than or equal to $\frac{v}{u'/4n} \le \frac{4nu'}{u'} = 4n$. By the proof of Theorem 4.1, the time complexity of one call to $R(u', 1/4)$ is $O(n^4)$. It is clear that the binary search in *RB* will stop after at most $O(\log n)$ calls of $R(u', 1/4)$, thus the total running time is bounded by $O(n^4 \log n)$.

Finally, to get an *FPTAS,* we need to run a slightly modified version of the algorithm *DP* with static interval partitioning. We describe this below.

Approximation Algorithm *ADP*

[*Initialization*] The same as that in the algorithm *DP.*

[*Partition*] Partition the interval $[\overline{v}, 2\overline{v}]$ into $\lceil n/\varepsilon \rceil$ equal intervals of size $\overline{v}\varepsilon/n$, with the last one possibly smaller.

[*Generation*] Generate set $S^{(k)}$ **for** $k = 1$ to $k = n + 1$ from $S^{(k-1)}$ as follows:

Set $\mathcal{T} = \emptyset$;

[*Operations*] The same as those in the algorithm *DP.*

[*Elimination*] Update set $S^{(k)}$.

1. If more than one state has a v value that falls into the same sub-interval, then discard all but one of these states, keeping only the *representative* state with the smallest t coordinate.
2. For any two states (k, b, t, d, v) and (k, b, t, d, v') with $v \le v'$, eliminate the one with v' from set $\mathcal{T}$ based on Remark 3.2;
3. Set $S^{(k)} = \mathcal{T}$.

Endfor

[*Result*] The best approximating solution corresponds to the state with the smallest v over all states in $\mathcal{T}^*$. Find the final schedule by backtracking through the ancestors of this state.

Theorem 4.4. *For any* $\varepsilon > 0$*, the algorithm ADP finds in* $O(n^4/\varepsilon)$ *time a schedule with cost v for the* $1|s,q|\sum_{j=1}^{n} w_j U_j + bq$ *problem, such that* $v \le (1 + \varepsilon)v^*$.

Proof. For each iteration in the algorithm *ADP,* the whole value interval $[\overline{v}, 2\overline{v}]$ is divided into subintervals with equal length $\frac{\varepsilon\overline{v}}{n}$ (the last subinterval may be less). Thus the maximum

number of the subintervals is less than or equal to $\frac{\bar{v}}{\varepsilon\bar{v}/n} = \frac{2n}{\varepsilon}$. By the proof of Theorem 3.1, the time complexity of the algorithm is $O(n^4/\varepsilon)$ indeed.

To summarize, the *FPTAS* applies the following algorithms to obtain an ε-approximation for the $1|s,q|\sum_{j=1}^{n} w_j U_j + bq$ problem.

The Fully Polynomial Time Approximation Scheme *(FPTAS)*

1. Run the algorithm *MW* by repeatedly calling *FC*(*k*) to determine $v' = w^* + q$;
2. Run the algorithm *RB* by repeatedly calling $R(u', 1/4)$ to determine $\overline{v}$;
3. Run the algorithm *ADP* using the bounds $\overline{v} \le v^* \le 2\overline{v}$.

Corollary 4.2. *The time and space complexity of the FPTAS is* $O(n^4 \log n + n^4/\varepsilon)$.

Proof. The time and space complexity follows from the proven complexity of the component algorithms.

5. Conclusions and further research

We presented a pseudopolynomial time dynamic programming algorithm for minimizing the sum of the weighted number of late jobs and the batch delivery cost on a single machine. For the special cases of equal weights or equal processing times, the algorithm *DP* requires polynomial time. We also developed an efficient, fully polynomial time approximation scheme for the problem.

One open question for further research is whether the algorithm *DP* and the *FPTAS* can be extended to the case of multiple customers.

6. References

P. Brucker and M.Y. Kovalyov. Single machine batch scheduling to minimize the weighted number of late jobs. *Mathematical Methods of Operation Research,* 43:1-8, 1996.

G.V. Gens and E.V. Levner. Discrete optimization problems and efficient approximate algorithms. *Engineering Cybernetics,* 17(6):1-11, 1979.

G.V. Gens and E.V. Levner. Fast approximation algorithm for job sequencing with deadlines. *Discrete Applied Mathematics,* 3(4):313-318, 1981.

R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.,* 4:287-326, 1979.

N.G. Hall and C.N. Potts. The coordination of scheduling and batch deliveries. *Annals Of Operations Research,* 135(1):41-64, 2005.

N.G. Hall. *Private communication*. 2006.

N.G. Hall and C.N. Potts. Supply chain scheduling: Batching and delivery. *Operations Research,* 51(4):566-584, 2003.

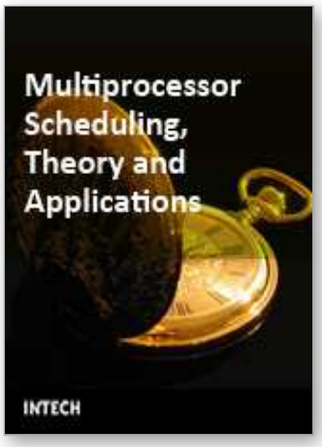
D.S. Hochbaum and D. Landy. Scheduling with batching: minimizing the weighted number of tardy jobs. *Operations Research Letters,* 16:79-86, 1994.

R.M. Karp. Reducibility among combinatorial problem. In R.E. Miller and Thatcher J.W., editors, *Complexity of Computer Computations,* pages 85-103. Plenum Press, New York, 1972.

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

S.K. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM,* 23(1): 116-127, 1976.

G. Steiner and R. Zhang. Minimizing the total weighted number of late jobs with late deliveries in two-level supply chains. *3rd Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA),* 2007.



Multiprocessor Scheduling, Theory and Applications

Edited by Eugene Levner

ISBN 978-3-902613-02-8

Hard cover, 436 pages

Publisher I-Tech Education and Publishing

Published online 01, December, 2007

Published in print edition December, 2007

A major goal of the book is to continue a good tradition - to bring together reputable researchers from different countries in order to provide a comprehensive coverage of advanced and modern topics in scheduling not yet reflected by other books. The virtual consortium of the authors has been created by using electronic exchanges; it comprises 50 authors from 18 different countries who have submitted 23 contributions to this collective product. In this sense, the volume can be added to a bookshelf with similar collective publications in scheduling, started by Coffman (1976) and successfully continued by Chretienne et al. (1995), Gutin and Punnen (2002), and Leung (2004). This volume contains four major parts that cover the following directions: the state of the art in theory and algorithms for classical and non-standard scheduling problems; new exact optimization algorithms, approximation algorithms with performance guarantees, heuristics and metaheuristics; novel models and approaches to scheduling; and, last but least, several real-life applications and case studies.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen