Minimizing the number of late jobs in case of stochastic processing times *

Marjan van den Akker
Department of Computer Science
Utrecht University
The Netherlands

Han Hoogeveen
Department of Computer Science
Utrecht University
The Netherlands

May 14, 2002

Abstract

We consider the single-machine scheduling problem of minimizing the number of late jobs. We first review and reinterpret the famous algorithm by Moore and Hodgson for the case of deterministic processing times as a dynamic programming algorithm. We then look at four problem classes with stochastic processing times. The first one has processing times that consist of a deterministic part and a random component that is independently, identically distributed for each job. The jobs in the other three classes have processing times that follow: (i) A gamma distribution with parameters α_j and β, where β is common to all jobs; (ii) A negative binomial distribution with parameters s_j and p, where p is the same for each job; (iii) A normal distribution with parameters μ_j and σ_j^2. In this scheduling environment, the completion times will be stochastic variables as well; under these circumstances, we qualify a job as being on time if the probability that it is completed by the deterministic due date is at least equal to a certain given minimum success probability. We show that for the first, second, and third class of instances the problem can be solved in $O(n \log n)$ time, where we need the additional assumption of equal minimum success probabilities in the first case. For the case with normally distributed processing times we present a pseudo-polynomial time algorithm, and we prove that this is the best we can hope for by establishing weak $\mathcal{NP}$-hardness. We also show that the problem of minimizing the weighted number of late jobs can be solved by an extension of the dynamic programming algorithm in all four cases; this takes pseudo-polynomial time. We further indicate how the problem of maximizing the expected number of on time jobs (with respect to the standard definition) can be tackled if we add the constraint that the on time jobs are sequenced in order of nondecreasing due dates.

Keywords. Scheduling, sequencing, single machine, number of late jobs, stochastic processing times, minimum success probability, dynamic programming, NP-hardness.

1 Introduction

One of the standard complaints of busy people nowadays is 'so many things to do, so little time'. If it is not possible to increase the amount of time available (for instance by hiring

*This work was partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

somebody), then it may be inevitable to skip some tasks. The question then becomes of course: which tasks should be skipped? This boils down to the machine scheduling problem that was studied in the famous paper by Moore (1968). The busy person is translated into a single machine, and the things to do are called jobs. Formally, the problem is then formulated as follows. The machine is assumed to be continuously available from time zero onwards, and it can perform at most one job at a time. The machine has to execute n jobs, denoted by $J_1, \ldots, J_n$. Performing task J_j requires a period of length p_j, and the execution of this task is preferably finished by its due date d_j. If job J_j is finished after its due date, then it is marked as late. The objective is to minimize the number of late jobs. Since it does not matter at which time a late job is finished, it can just as well be skipped altogether; the machine then only carries out the jobs that will finish on time. A real-life example is a company that receives requests for service: each request leads to a job with a certain processing time and due date; the firm has to tell the potential client whether it will honor its request. If the request is denied, then the job will not be executed. Moore shows that the problem of maximizing the number of jobs that are finished on time can be solved in $O(n \log n)$ time by an algorithm that since then is known as Moore-Hodgson's algorithm.

In this setting, each job is equally important. In many applications, however, some jobs are more important than others. This importance can be measured by assigning a positive weight w_j to each job J_j ($j = 1, \ldots, n$); the objective function then becomes to minimize the total weight of the late jobs. Lawler and Moore (1969) show that this problem is solvable in $O(n \sum p_j)$ time by dynamic programming. Karp (1972) shows that pseudo-polynomial running time is unavoidable for this problem (unless $\mathcal{P} = \mathcal{NP}$) by establishing $\mathcal{NP}$-hardness in the ordinary sense, even if all due dates are equal.

In this paper, we look at the problems described above, but we abolish one of the main assumptions of scheduling theory, which is that the data are deterministic. In our case, we assume stochastic processing times; additionally, we assume that the processing times are independent of each other. We consider four specific classes of instances. In the first one the processing times are stochastic variables that are distributed according to a gamma distribution with parameters α_j (which varies per job) and β (which is equal for all jobs). The gamma distribution is often applied to model the processing time of a task (see for instance Law and Kelton, 2000). The second class of processing times is used to model a production process where items are produced that work well with probability p and malfunction with probability $(1-p)$; a job J_j corresponds then to an order of s_j correctly functioning items. The corresponding processing time then follows a negative binomial distribution with parameters s_j and p. In the third class the processing times consist of a deterministic component and a random disturbance, which we assume to be identically distributed for each job. This can be used to model the situation that the disturbances in the production process are not job-related but due to some side-equipment that is used by each job in the same way. In the last case, we assume that the processing times follow a normal distribution with known expected value p_j and known variance σ_j^2.

We suppose that each due date is deterministic, which is reasonable, as they are specified by the customer issuing the request. More importantly, we further assume that this customer is willing to accept a delayed completion of his order, as long as the probability that the order is on time is at least equal to some given lower bound value, which we define as the *minimum success probability*.

The remainder of the paper is organized as follows. In Section 2 we review the problem of minimizing the number of late jobs with deterministic processing times, and we explain

Moore-Hodgson's algorithm as a dynamic programming algorithm. In Section 3 we discuss the consequences of working with stochastic processing times. We show that the first and second class of processing times can be reformulated as deterministic problems, and hence can be dealt with by the traditional algorithms. For the other two classes of processing times, we need the additional assumption that the minimum success probabilities are all equal. We develop dynamic programming algorithms that minimize the (weighted) number of late jobs for these instances; here we heavily use the insight gained in Section 2. We further discuss the problems that we face in case of general minimum success probabilities. In Section 4, we show that the problem with stochastic processing times that follow a normal distribution is fundamentally more difficult than the problem with deterministic processing times by establishing ordinary $\mathcal{NP}$-hardness for the problem of minimizing the number of late jobs. In Section 5, we address the problem of minimizing the expected value of the (weighted) number of late jobs under the side-constraint that the on time jobs are executed in EDD-order. Finally, we draw some conclusions in Section 6.

2 Moore-Hodgson's algorithm reviewed

In this section, we take a closer look at the problem with deterministic processing times; we need this insight in Section 3. We start with some simple, well-known observations. As mentioned before, the completion time of job J_j, which we denote by C_j, has become unimportant once it is greater than the due date. Hence, the late jobs are executed after all the on time jobs (if the late jobs are executed at all). The second observation is that the jobs that are marked as on time are executed in order of nondecreasing due date, which is known as the EDD-order; this is due to Jackson (1955), who showed that executing the jobs in EDD-order minimizes the maximum lateness, where the lateness of a job j is defined as $C_j - d_j$ (this maximum lateness is at most equal to zero if all jobs are on time). Hence, instead of specifying the whole schedule, we can limit ourselves to specifying the set E (for early) containing the on time jobs, as it can be checked whether the jobs in E can be all on time together by putting them in EDD-order. A set E of jobs that are selected to be on time is called *feasible* if none of them is late when executed in EDD-order. The goal is therefore to find a feasible set E of maximum cardinality. From now on, we will use the following notation. By $|Q|$ and $p(Q)$ we denote the number and total processing time of the jobs in a given set Q, respectively. Since the EDD-order is crucial in the design of the algorithm, we assume from now on that the jobs are numbered such that

$$d_1 \leq d_2 \leq \ldots \leq d_n.$$

We want to show that Moore-Hodgson's algorithm is in fact a dynamic programming algorithm with a special structure; we need this in Section 3. For this dynamic programming algorithm, the following *dominance rule* is crucial.

Dominance Rule 2.1 *Let E^1 and E^2 be two feasible subsets of the tasks $\{J_1, \ldots, J_j\}$ with $|E^1| = |E^2|$. If $p(E^1) < p(E^2)$, then any solution E with $E \cap \{J_1, \ldots, J_j\} = E^2$ can be ignored.*

Proof. Let E correspond to an optimal solution, and suppose that $E \cap \{J_1, \ldots, J_j\} = E^2$. We will show that replacing the jobs in $E \cap E^2$ by the jobs in E^1 yields a feasible subset $\bar{E}$ of $\{J_1, \ldots, J_n\}$; since $|\bar{E}| = |E|$, the subset $\bar{E}$ must then correspond to an optimal solution, too.

To show that $\bar{E}$ is a feasible subset, we must show that in the EDD-schedule of $\bar{E}$ all jobs are on time. Let σ and π denote the EDD-schedule of the jobs in $\bar{E}$ and E, respectively. Due to the numbering of the jobs, we know that the jobs in E^1 precede the remaining jobs of $\bar{E}$ in σ; as E^1 is a feasible subset, these jobs are on time in σ. The remaining jobs in $\bar{E}$ start $p(E^2) - p(E^1) > 0$ time units earlier in σ than in π, and hence these jobs are on time as well. ∎

As a consequence of the dominance rule, the only feasible subset of $\{J_1, \ldots, J_j\}$ with cardinality k that we have to care about is the one with minimum total processing time. We define z_j $(j = 1, \ldots, n)$ as the maximum number of jobs in the set $\{J_1, \ldots, J_j\}$ that can be on time together; the value of z_j will be determined through the algorithm. We further define $E_j^*(k)$ $(j = 1, \ldots, n; k = 0, \ldots, z_j)$ as the feasible subset containing exactly k jobs from $\{J_1, \ldots, J_j\}$ with minimum total processing time. We derive a dynamic programming algorithm to solve the problem of minimizing the number of late jobs as follows. We add the jobs one by one in EDD-order. For each combination (j, k), where j $(j = 1, \ldots, n)$ refers to the number of jobs that have been considered and k $(k = 0, \ldots, z_j)$ denotes the number of on time jobs, we introduce a statevariable $f_j(k)$ with value equal to $p(E_j^*(k))$. As an initialization, we define $z_0 = 0$ and put $f_j(k) = 0$ if $j = k = 0$ and $f_j(k) = \infty$, otherwise. Suppose that we have determined the values $f_j(k)$ for a given j and all $k = 0, \ldots, z_j$. We then can determine z_{j+1} by checking whether $f_j(z_j) + p_{j+1} \leq d_{j+1}$: if this is the case, then $z_{j+1} = z_j + 1$, and $z_{j+1} = z_j$ otherwise. Now we put $f_{j+1}(0) = 0$ and determine $f_{j+1}(k)$ $(k = 1, \ldots, z_{j+1})$ through the recurrence relation

$$f_{j+1}(k) = \min\{f_j(k), f_j(k-1) + p_{j+1}\}.$$

Assuming the correctness of our approach (which we shall show below), we can compute the values $f_j(k)$ in $O(n^2)$ time altogether, from which we immediately determine the minimum number of late jobs as $(n - z_n)$, whereas the optimum schedule can be determined through backtracking.

Lemma 2.2 *The value $f_j(k)$ computed through the recurrence relation is equal to the minimum total processing time of all feasible subsets of $\{J_1, \ldots, J_j\}$ containing k jobs.*

Proof. Because of the intialization, we find $f_j(k) = \infty$ if $k > z_j$, which is the correct value by definition. We use induction to prove the correctness for all j and $k \leq z_j$. For $j = 0$ and $k = 0$, correctness holds because of the initialization. Suppose that the values $f_j(k)$ are correct for all $j = 0, \ldots, m$ and $k = 0, \ldots, z_m$ for some value $m \geq 0$. We will show correctness of $f_{m+1}(k)$, where k is any value from $\{1, \ldots, z_{m+1}\}$ (the case $k = 0$ being trivial). If $k \leq z_m$, then there are two possibilities: we add job J_{m+1} to the set of on time jobs, or we mark it as late. In the latter case, we find a minimum total processing time of $f_m(k)$. If we include J_{m+1} in the set of on time jobs, then we have to select a feasible subset from the jobs $\{J_1, \ldots, J_m\}$ containing $k - 1$ jobs, and we know that the best one has total processing time $f_m(k-1)$. Job J_{m+1} does not necessarily have to be on time when placed after this subset of $k - 1$ jobs, but it will be if $f_m(k-1) + p_{m+1} < f_m(k)$, since $f_m(k)$ is equal to the completion time of the last job in the set of on time jobs, which amounts to no more than $d_m \leq d_{m+1}$ by definition.

If $k = z_{m+1} = z_m + 1$, then we need to add job J_{m+1} to the set of on time jobs to get the desired number. In that case, the check performed when computing z_{m+1} guarantees that J_{m+1} is on time when it starts at time $f_m(z_m)$, and the correctness of the value $f_m(z_m)$ settles the proof of the correctness of $f_{m+1}(z_{m+1})$. ∎

This algorithm immediately carries over to the weighted case, since the dominance rule holds true (with total weight of the on time jobs instead of the cardinality of the set of on time jobs). The statevariables $f_j(k)$ are defined as before, but k then assumes the values $0, 1, \ldots, \sum_{i=1}^{j} w_i$. Hence, the running time of the dynamic programming algorithm becomes $O(n \sum w_i)$. Note that we have to change the recurrence relation to

$$f_{j+1}(k) = \min\{f_j(k), f_j(k - w_{j+1}) + p_{j+1}\}.$$

Now we return to the case with unit weights. We will show that by looking at the dynamic programming algorithm in the right way we can obtain Moore and Hodgson's algorithm. Here we need the close relation between the sets $E_j^*(k)$ and $E_j^*(k-1)$ expressed in the following lemma.

Lemma 2.3 *The set $E_j^*(k-1)$ is obtained from the set $E_j^*(k)$ by removing a task from $E_j^*(k)$ with largest processing time.*

Proof. Suppose that the lemma does not hold for some combination of j and k. Let J_i be a task in $E_j^*(k)$ with maximum processing time. According to our assumption, $p(E_j^*(k-1)) < p(E_j^*(k)) - p_i$. Determine J_q as the job with maximum due date that belongs to $E_j^*(k)$ but not to $E_j^*(k-1)$. Now consider the subset $\bar{E}_j^*(k) = E_j^*(k-1) \cup \{J_q\}$; we will prove that this subset is feasible, which contradicts our assumption, since $p(E_j^*(k)) > p(E_j^*(k-1)) + p_i \geq p(E_j^*(k-1)) + p_q = p(\bar{E}_j^*(k))$. We have to check the EDD-schedule for the tasks in $\bar{E}_j^*(k)$, which is obtained from the EDD-schedule for the jobs in $E_j^*(k-1)$ by inserting J_q at the correct spot. Due to the feasibility of $E_j^*(k-1)$, the only jobs that may be late are J_q and the jobs after J_q. But all these jobs belong to $E_j^*(k)$ by choice of J_q and in comparison to the EDD-schedule of $E_j^*(k)$ they are started $p(E_j^*(k)) - p(\bar{E}_j^*(k)) > 0$ time units earlier. Since $E_j^*(k)$ is a feasible subset, the same holds for $\bar{E}_j^*(k)$. ∎

As a consequence, if we know the *set* $E_j^*(z_j)$ (and not just its total processing time), then we can determine the set $E_{j+1}^*(z_{j+1})$ by applying the recurrence relation, where we can compute $p(E_j^*(z_j))$ (and $p(E_j^*(z_j - 1))$ if necessary) directly from $E_j^*(z_j)$. The recursion step is then:

- If $p(E_j^*(z_j)) + p_{j+1} \leq d_{j+1}$, then $E_{j+1}^*(z_{j+1}) \leftarrow E_j^*(z_j) \cup \{J_{j+1}\}$.
- If $p(E_j^*(z_j)) + p_{j+1} > d_{j+1}$, then find the longest job in $E_j^*(z_j) \cup \{J_{j+1}\}$ and set $E_{j+1}^*(z_{j+1})$ equal to $E_j^*(z_j) \cup \{J_{j+1}\}$ minus this job.

But this leads exactly to Moore and Hodgson's algorithm, which is defined as follows:

MOORE & HODGSON

STEP 1. Set σ equal to the EDD-schedule for all jobs $J_1, \ldots, J_n$.

STEP 2. If each job in σ is on time, then stop.

STEP 3. Find the first job in σ that is late; let this be job J_j. Find the largest job from the set containing J_j and all its predecessors in σ and remove it from σ. Remove the idle time from the resulting schedule by shifting jobs forward; call this new schedule σ, and go to STEP 2.

3 Stochastic processing times

From now on we assume that the processing times are stochastic variables, which we denote by π_j ($j = 1, \ldots, n$); we stick to deterministic due dates, which is not unreasonable, as they are specified by the costumers. Since the processing times are no longer fixed, we cannot represent a schedule anymore by specifying the time intervals during which each job is processed. Therefore, we represent the schedule by specifying the order in which the jobs are to be scheduled; when running the schedule we determine the starting and completion times through the 'wait-and-see' approach: as soon as a job is finished, we start the next one in the sequence. Note that the decision of specifying whether a job is marked as on time must be made before the start of the schedule; we cannot await the actual processing times of the first couple of jobs in the schedule and use this information in our decision of marking future jobs on time or tardy. As a consequence, according to the old definition, we can then only classify a job as on time or late after the schedule has been executed. To compute the quality of a schedule beforehand, we change our definition of on time as follows.

Definition 3.1 *Given the sequence in which the jobs are to be executed, a job is considered to be on time if the probability that it is completed by its due date is at least equal to some given probability, which we call the minimum success probability.*

The objective function is then to minimize the number of late jobs. We will consider four classes of instances of stochastic processing times; in all cases we assume that the corresponding stochastic variables are independent. The four classes are defined as follows:

- The processing time π_j of job J_j ($j = 1, \ldots, n$) follows a gamma distribution with parameters p_j (we use p_j instead of α_j, which is commonly used to denote this parameter) and β, which is equal for all jobs. The expected processing time is then equal to $p_j\beta$. Since the scale parameter β is equal for all jobs, the total processing time $\sum_{i\in S}\pi_i$ follows a gamma distribution with parameters $\sum_{i\in S} p_i$ and β.

- The processing time π_j of job J_j ($j = 1, \ldots, n$) is distributed according to a negative binomial distribution with parameters p_j (we replace the commonly used s_j by p_j) and r (we use r instead of p to avoid possible confusion with p_j), which is equal for all jobs. The expected processing time is then equal to $p_j(1-r)/r$. As r is equal for each job, the total processing time $\sum_{i\in S}\pi_i$ follows a negative binomial distribution with parameters $\sum_{i\in S} p_i$ and r.

- The processing time π_j of job J_j ($j = 1, \ldots, n$) consists of a deterministic part p_j and a random disturbance, where the disturbances are independently, identically distributed random variables.

- The processing time π_j of job J_j ($j = 1, \ldots, n$) follows a normal distribution with known expected value p_j (we use p_j instead of μ_j) and known variance σ_j^2. Hence, $\sum_{i\in S}\pi_i$ follows a normal distribution with expected value $\sum_{i\in S} p_i$ and variance $\sum_{i\in S}\sigma_i^2$.

We will show in Subsection 3.1 that the first two classes in fact boil down to deterministic problems. For the other two classes require a more intricate approach, where we use the insight gained in the previous section to solve these problems through dynamic programming; this is described in the Subsections 3.2 and 3.3.

3.1 Gamma and negative binomial distributed processing times

We first describe our solution approach for the class of Gamma distributed processing times; this approach can be immediately carried over to the class of negative binomial distributed processing times.

As we have remarked above, the completion time of job J_j is equal to the total processing time of the jobs in the set S, which contains job J_j and all its predecessors. Hence, $C_j = \sum_{i \in S} \pi_i$ follows a gamma distribution with parameters $\sum_{i \in S} p_i$ and β. If we know $p(S) = \sum_{i \in S} p_i$, then we can easily compute $P[C_j \leq d_j]$, as β and d_j are known parameters. Given the minimum success probability of job J_j, which we denote by y_j, we must compare $P[C_j \leq d_j]$ to y_j, which results in a qualification of job J_j as on time or tardy. But since $P[C_j \leq d_j]$ is a monotone function of $p(S)$, we can easily compute D_j as the largest integral value of $p(S)$ such that $P[C_j \leq d_j] \geq y_j$ if C_j follows a gamma distribution with paramaters $p(S)$ and β, for instance by applying binary search.

Suppose that we have computed the value D_j for each $j = 1, \ldots, n$. But then we have converted our stochastic problem into a problem with jobs that are described by deterministic values p_j and deterministic values D_j, which decree an upper bound on $p(S)$. But this is exactly equal to the setting of the original, deterministic problem, with processing times p_j and due dates D_j. Hence, we can solve this problem to optimality by applying Moore-Hodgson's algorithm in $O(n \log n)$ time, once we have determined the values D_j $(j = 1, \ldots, n)$.

If we look at the weighted problem, then we can again determine the adjusted due dates D_j, as these do not depend on the weights. Hence, we can solve the weighted case in time $O(n \sum p_j)$ through the algorithm of Lawler and Moore or in time $O(n \sum w_j)$ through the dynamic programming algorithm of Section 2, given the values D_j $(j = 1, \ldots, n)$. By applying Karp's reduction, we can show that the weighted case is $\mathcal{NP}$-hard in the ordinary sense.

Note that the on time jobs are executed according to the order of the adjusted due dates D_j, which does not have to coincide with the order of the original due dates d_j, as the minimum success probabilities do not have to be equal.

Now we come to the class of negative binomial distributed processing times. It is easily verified that we can deal with this in the same way as the distribution of the completion time of job J_j is fully determined by $\sum_{i \in S} p_i$ and the given parameter r. Hence, given the values D_j $(j = 1, \ldots, n)$ the unweighted problem is solved in $O(n \log n)$, and the weighted case is solved in time $O(n \sum p_j)$ or $O(n \sum w_j)$.

Note that the same approach can be used for the normal distribution in the special case that for each job the variance is equal to some constant times the expected value. In Subsection 3.3 we deal with the general case of this problem.

3.2 Equally disturbed processing times

Again, we use C_j to denote the stochastic completion time of J_j $(j = 1, \ldots, n)$, and we let S denote the set containing all jobs that have been selected to complete on time up to and including J_j. Hence, we have that $C_j = \sum_{i \in S} \pi_i$, which consists of $p(S)$ and the joint disturbance of the jobs in S. This implies, however, that we cannot determine D_j like we did in the previous subsection, since this value now depends on $|S|$ as well. Hence, we return to the basics of our dynamic programming approach.

The crux behind the dynamic programming algorithm of Section 2 was that a fixed order was known in which the jobs could be added without running the risk of missing the optimum, namely the EDD-order based on the d_j values. As we do not know the order in which the jobs must be added in case of arbitrary y_j values, we make the additional assumption that the minimum success probabilities are equal. In Theorem 3.2 below we show that adding the jobs in EDD-order then enables us to find an optimum solution. We discuss the problem of what to do in case of arbitrary minimum success probabilities in Subsection 3.4.

Since all disturbances are independently, identically distributed, the composite disturbance only depends on the number of jobs that belong to S. Hence, to compute the probability $P[C_j \leq d_j]$, which has to be compared to the minimum success probability, we need to know $p(S)$ and $|S|$. Obviously, $P[C_j \leq d_j]$ is maximized by choosing for $S \setminus \{J_j\}$ the feasible subset of $\{J_1, \ldots, J_{j-1}\}$ of minimum total deterministic processing time from among the sets with equal cardinality. Therefore, the required knowledge to compute $P[C_j \leq d_j]$ is captured by the information stored in the statevariables $f_j(k)$ of the previous section. Hence, we can apply the analysis of the previous section, which leads to Moore-Hodgson's algorithm. Note that in the algorithm the definition of J_{j+1} being on time has to be modified, but this is a piece of cake. What is left to prove is the validity of the EDD-order.

Theorem 3.2 *If the minimum success probability is equal for all jobs, then there exists an optimal sequence in which the on time jobs are in EDD-order.*

Proof. Suppose to the contrary that the optimal sequence does not schedule the jobs in EDD-order. Then there must be two adjacent on time jobs, say J_i and J_j, where J_i precedes J_j although $d_i > d_j$. If we interchange the two, then J_j is completed earlier, which implies that it is on time again. Moreover, after the interchange the completion time C_i has equal distribution as the completion time C_j before the interchange, and as $d_i > d_j$ we find that the probability that J_i meets its due date after the interchange is at least equal to the probability that J_j met its due date before the interchange, which is known to be no less than the minimum success probability. Hence, we find through a repetition of this argument that resequencing the set of on time jobs in EDD-order leaves these jobs on time. ■

The proof above also holds if the minimum success probabilities form a nonincreasing sequence when the jobs are put in EDD-order (we come back to this issue in Subsection 3.4, where we give a formal proof of this statement in Theorem 3.3). For the ease of presentation, however, we keep the minimum success probability constant, but each result obtained in the remainder of the paper also holds for these nonincreasing minimum success probabilities.

It is easy to solve the weighted case by extending the statevariables $f_j(k)$ to contain as extra information the total weight of the current set of on time jobs (where the recursion is modified in an appropriate fashion). In contrast to the deterministic case in the previous section, however, we now need both the total weight of the current set of on time jobs and the cardinality of this set (the latter for computing $P[C_j \leq d_j]$). Hence, the algorithm will then run in $O(n^2 \sum w_i)$ time. Through a reduction like the one proposed by Karp to show $\mathcal{NP}$-hardness of the deterministic problem (which in fact is a problem that belongs to this class) it is easily verified that pseudo-polynomial running time is the best we can hope for in this case.

3.3 Normally distributed processing times

In this subsection we assume that the processing time π_j of job J_j $(j = 1, \ldots, n)$ is a normally distributed random variable with expected value p_j and variance σ_j^2. We further assume that all π_j variables are independent of each other. Since the variances can differ per job, we cannot apply the analysis of the previous subsections. Again, we define C_j as the stochastic completion time of J_j ($j = 1, \ldots, n$), and we let S denote the set containing all jobs that have been selected to complete on time up to and including J_j. Hence, we have that $C_j = \sum_{i \in S} \pi_i$ follows the normal distribution with mean $\sum_{i \in S} p_i$ and variance $\sum_{i \in S} \sigma_i^2$. If these two quantities are known, then we can immediately compute the probability $P[C_j \leq d_j]$ and compare this to the minimum success probability, which from now on we denote by y. Again, Theorem 3.2 shows that there exists an optimal schedule that starts with the on time jobs in EDD-order.

Obviously, not all possible combinations $(\sum_{i \in S} p_i, \sum_{i \in S} \sigma_i^2)$ are of interest. If we look at all feasible subsets S of $\{J_1, \ldots, J_j\}$ with equal $\sum_{i \in S} \sigma_i^2$ value and equal cardinality, then our best chance to meet y is offered by the subset with minimum $\sum_{i \in S} p_i$ value. On the other hand, if we can choose among all feasible subsets S of $\{J_1, \ldots, J_j\}$ with equal $\sum_{i \in S} p_i$ value and equal cardinality, then we would like to have the one with smallest $\sum_{i \in S} \sigma_i^2$ value if $y \geq 0.5$ and the one with maximum $\sum_{i \in S} \sigma_i^2$ value if $y < 0.5$. Since it sounds reasonable to assume that $y \geq 0.5$, we work out the dynamic programming algorithm by determining for each value $t \in \{0, 1, \ldots, \sum_{i=1}^{j} p_j\}$ and for each cardinality $k = 0, \ldots, z_j$ the feasible subset S with minimum total variance, if it exists. The reason behind enumerating the total processing time instead of the total variance is that presumably $\sum_{i=1}^{n} \sigma_i^2 > \sum_{i=1}^{n} p_i$, which is advantageous for the running time of the algorithm. If this is not the case, or if $y < 0.5$, then we should include the variance in the statevariable, which gets a value equal to the total processing time.

To get the dynamic programming algorithm running, we introduce for each $j = 1, \ldots, n$; $k = 1, \ldots, z_j$; and $t = 0, 1, \ldots, \sum_{i=1}^{j} p_j$ the statevariable $f_j(k, t)$, which is supposed to denote the minimum variance $\sum_{i \in S} \sigma_i^2$ over all feasible subsets S of $\{J_1, \ldots, J_j\}$ with $|S| = k$ and $\sum_{i \in S} p_i = t$. Remark that z_j is defined as the maximum cardinality of any feasible subset S of $\{J_1, \ldots, J_j\}$. As our initialization, we put $z_0 = 0$ and define $f_j(k, t) = 0$ if $j = k = t = 0$ and ∞, otherwise. As before, we add the jobs in EDD-order. When adding job J_{j+1}, we first determine z_{j+1}. Thereto, we check for each $f_j(z_j, t)$ whether

$$P\left(z \leq \frac{d_{j+1} - t - p_{j+1}}{\sqrt{f_j(z_j, t) + \sigma_{j+1}^2}}\right) \geq y,$$

where z denotes a standard normal variable. If this holds for none of the statevariables $f_j(z_j, t)$, then we put $z_{j+1} = z_j$; otherwise, we put $z_{j+1} = z_j + 1$. We then put $f_{j+1}(0, 0) = 0$ and compute $f_{j+1}(k, t)$ for $k = 1, \ldots, z_{j+1}$ and $t = 0, \ldots, \sum_{i=1}^{j+1} p_j$ through the following recurrence relation

$$f_{j+1}(k, t) = \min\{f_j(k, t), f_j(k - 1, t - p_{j+1}) + \sigma_{j+1}^2\}.$$

The first term corresponds to including job J_{j+1} in the set of late jobs, whereas job J_{j+1} is included in the on time set in the second case. The optimum value of the objective function is determined as $n - z_n$, and the corresponding on time set can be found through backtracking. This algorithm has running time $O(n^2 \sum p_i)$.

The weighted case can be easily dealt with by a clone of the above algorithm. In that case, we let k refer to the total weight of the on time jobs; we do not need the cardinality of the on time set anymore. Hence, the algorithm then runs in $O(n \sum p_i \sum w_i)$ time.

3.4 Challenges when dealing with unequal minimum success probabilities

Suppose that the minimum success probabilities are not all equal; as before, we denote the minimum success probability of job J_j by y_j, for $j = 1, \ldots, n$. We start with a two-job example, which belongs to both class three and four, that shows that the EDD-order is not necessarily optimal then. Both jobs have a processing time that follows a normal distribution with $p_1 = 12$ and $p_2 = 8$ and $\sigma_1^2 = \sigma_2^2 = 1$. The due dates are $d_1 = 20$ and $d_2 = 21$, and the minimum success probabilities are $y_1 = 0.5$ and $y_2 = 0.95$. A straightforward calculation shows that in the EDD-sequence job J_2 is late, whereas in the reverse sequence both jobs are on time.

This is bad news of course. On the bright side, for both our special classes it is possible to check in $O(n^2)$ time whether all jobs in a given set can be on time simultaneously, even if the minimum success probabilities are unequal, through the following rule, which resembles Lawler's *Least-cost-last-rule* (Lawler, 1973): compute the probability distribution of the, yet unknown, last job in the sequence, and determine whether there exists a job that can be sequenced last without violating its minimum success probability. If there is no such job, then clearly this set is not feasible. Otherwise, select any such job, sequence it last, and repeat the procedure for the remaining jobs. The correctness of this rule is obvious, and therefore we state it without proof.

Since we are able to check for a given job set whether it is feasible, one approach might be to use the dynamic programming algorithm of the previous subsections, where we use the above algorithm to check whether adding the current job to a feasible subset yields a feasible subset again. Unfortunately, this approach does not work, as follows from the following three-job example: we add job J_0 to our two-job example with $p_0 = 11$, $\sigma_0^2 = 1$, $d_0 = 15$, and $y_0 = 0.5$. A quick computation shows that the only possibility of having two jobs on time is to put J_2 first and J_1 second. But when we apply our dynamic programming algorithm where we add the jobs in EDD-order, then this solution is eliminated, as the feasible subset of cardinality one from $\{J_0, J_1\}$ with minimum processing time contains J_0 and not J_1. Note that the dynamic programming algorithm works fine if we add the jobs in the order J_0, J_2, J_1.

Hence, the question now becomes: in which order should we add the jobs to the dynamic programming algorithm? Finding an optimal one, that is, an order which leads to the optimal solution, is a challenging open question. Below, we give a partial description of an optimal order for adding the jobs to the dynamic programming algorithm. If the number of possibly optimal orders is not too big, then one approach is to try them all and pick the best solution.

Theorem 3.3 *If $d_i \leq d_j$ and $y_i \leq y_j$, then there exists an optimal order for adding the jobs to the dynamic programming algorithm in which J_i precedes J_j.*

Proof. Consider the optimal solution (scheduling order). If we add the on time jobs in this order to the dynamic programming solution with the tardy jobs interleaved, then the dynamic programming algorithm will come up with this solution or an equivalent on. Hence, it suffices to show that there exists an optimum scheduling order in which J_i precedes J_j if both jobs are on time. We prove this by a contradictory argument. If J_j precedes J_i in this scheduling order, then we modify it by removing J_j from its current position and reinserting it immediately after J_i. Each job affected by this move goes forward in the scheduling order (and hence remains on time), except for J_j, which gets some more predecessors. The completion time of J_j is equally distributed as the completion time of J_i in the original scheduling order, and J_i

was on time. Since $d_i \leq d_j$ and $y_i \leq y_j$, job J_j must be on time then, too, which implies that this new scheduling order is at least as good as the previous one. ■

4 $\mathcal{NP}$-hardness

In this section we show that the problem with normally distributed processing times is fundamentally more difficult than the problem with deterministic processing times, which is solved in polynomial time. We show that the problem is $\mathcal{NP}$-hard in the ordinary sense through a reduction from the problem PARTITION, which is known to be $\mathcal{NP}$-complete in the ordinary sense. PARTITION is defined as follows:

PARTITION
Given t positive integers $a_1, a_2, \ldots, a_t$ with sum equal to $2A$, does there exist a subset S of the index set $\{1, \ldots, t\}$ such that

$$\sum_{i \in S} a_i = A?$$

Given any instance of PARTITION, we construct the following instance of the decision variant of our problem. Each a_i ($i = 1, \ldots, t$) leads to two jobs: J_{2i-1} and J_{2i}. Moreover, there is a special job J_{2t+1}. The data are found in the table below, where the index i runs from 1 to t. The symbols B, M, and Q stand for numbers. B is such that $B(\sqrt{A+1} - \sqrt{A}) > A$; a straightforward computation shows that putting B equal to $2A\sqrt{A+1}$ rounded down does the trick. The integer M is defined as $A + 1 + B\sqrt{A}$ rounded down. The integer Q is chosen such that $\sqrt{Q + B^2 A} - \sqrt{Q} < 1$; a straightforward computation shows that $Q = A^6(A+1)^2$ is sufficient. The minimum success probability y is set equal to 0.8413; this choice is motivated by the fact that $P(z \leq 1) = 0.8413$ for a standard normal variable z.

	p_j	σ_j^2	d_j
J_{2i-1}	iM	$B^2 a_i$	$\sum_{h=1}^{i} hM + A + B\sqrt{A}$
J_{2i}	$iM + a_i$	0	$\sum_{h=1}^{i} hM + A + B\sqrt{A}$
J_{2t+1}	(t+1)M	Q	$\sum_{h=1}^{t+1} hM + A + \sqrt{Q + B^2 A}$

Table 1: Data for our instance

The decision variant of our problem is defined as the following question: does the instance defined above have a solution in which no more than t jobs are late?

We will show that the answer to PARTITION is 'yes' if and only if the decision problem is answered affirmatively. The idea behind the reduction is that at most t of the first $2t$ jobs can be on time, and we need all of them to get a 'yes' to the decision variant. Define the index set S such that $j \in S$ if and only if J_{2j} belongs to the set of on time jobs, for $j = 1, \ldots, t$. To let either J_{2t-1} or J_{2t} finish on time, we need that $\sum_{j \in S} a_j \geq A$. Then J_{2t+1} comes into play; the instance is chosen such that J_{2t+1} can be on time only if $\sum_{j \in S} a_j \leq A$.

Due to the choice of the minimum success probability, a job J_j ($j = 1, \ldots, 2t+1$) is on time if the expected value of C_j plus the standard deviation (which is the square root of the variance) of C_j is no more than its due date.

First suppose that the answer to PARTITION is 'yes'; let S denote the subset of $\{1, \ldots, t\}$ with $\sum_{j \in S} a_j = A$. Then we construct a 'yes' solution to the decision variant as follows. We construct the set E of on time jobs such that it includes J_{2j-1} if $j \notin S$ and J_{2j} if $j \in S$, for

$(j = 1, \ldots, t)$; the last job in E is job J_{2t+1}. Let $J_{[i]}$ $(i = 1, \ldots, t+1)$ denote the i'th job in E when ordered according to the EDD-rule; define $C_{[i]}$, $p_{[i]}$, and $d_{[i]}$ accordingly. It is easily verified that the expected value of $C_{[i]}$ is no more than $\sum_{h=1}^{i} hM + A$ for $i = 1, \ldots, t+1$, and that the variance of $C_{[i]}$ is at most equal to $B^2 A$, for $i = 1, \ldots, t$, whereas $C_{[t+1]}$ has variance $Q + B^2 A$. Hence, all jobs in E finish on time, which implies that no more than t jobs are late, which means 'yes'.

Conversely, suppose that the answer to the decision variant is 'yes'. Let E denote the set of on time jobs. We first show that then exactly one of each pair of jobs $\{J_{2i-1}, J_{2i}\}$ $(i = 1, \ldots, t)$ must belong to E; hence, J_{2t+1} must then also be on time to get the count right. To facilitate the proof, we put $p_{2i} = iM$ and $\sigma_{2i-1}^2 = 0$ for $i = 1, \ldots, t$; if we can prove that $J_{[i]} \in \{J_{2i-1}, J_{2i}\}$ $(i = 1, \ldots, t)$ for this instance, then it automatically holds for the original instance. It is easily verified, for instance by applying Moore-Hodgson's algorithm, that at most i $(i = 1, \ldots, t)$ of the jobs $J_1, \ldots, J_{2i}$ can be on time, and that the minimum total processing time of such a subset is equal to $\sum_{h=1}^{i} hM$. Now look at the jobs J_{2t-1}, J_{2t}. Since at most $t-1$ of the jobs from $J_1, \ldots, J_{2t-2}$ can be on time, at least one of the jobs J_{2t-1}, J_{2t} must belong to E. If E contains both, then E contains $t-2$ jobs from $\{J_1, \ldots, J_{2t-2}\}$ with at most one job from each pair $\{J_{2i-1}, J_{2i}\}$. Hence, the jobs in E then have a total processing time of at least $\sum_{h=1}^{t-2} hM + 2tM$. But then

$$p(E) \geq \sum_{h=1}^{t-2} hM + 2tM = \sum_{h=1}^{t} hM + M > \sum_{h=1}^{t} hM + A + B\sqrt{A} = \max\{d_{2t-1}, d_{2t}\},$$

which shows that such a set E cannot be feasible. Hence, E contains exactly one job from J_{2t-1}, J_{2t}. We repeat this argument for J_{2t-3}, J_{2t-2}, which shows that E must contain exactly one job from J_{2t-3}, J_{2t-2}, etc.

We define S as the subset of $\{1, \ldots, t\}$ that contains index i if and only if J_{2i} belongs to the on time set. Let $J_{[t]}$ denote the job from the pair $\{J_{2t-1}, J_{2t}\}$ that is on time. The expected value of $C_{[t]}$ is equal to $\sum_{h=1}^{t} hM + \sum_{j \in S} a_j$, and the variance of $C_{[t]}$ is equal to $B^2 \sum_{j \notin S} a_j$. If $\sum_{j \in S} a_j < A$, then the variance of $C_{[t]}$ is at least equal to $B^2(A+1)$. Hence, the expected value of $C_{[t]}$ plus the standard deviation of $C_{[t]}$ is then at least equal to

$$\sum_{h=1}^{t} hM + B\sqrt{A+1} > \sum_{h=1}^{t} hM + A + B\sqrt{A} = d_{[t]},$$

where the '>' sign comes from the choice of B. This result disqualifies job $J_{[t]}$ as on time, and hence we must have $\sum_{j \in S} a_j \geq A$.

From now on, we use $a(S)$ as a short-hand notation for $\sum_{j \in S} a_j$. Now we look at job J_{2t+1}, which is on time as well. The expected value of C_{2t+1} is equal to $\sum_{h=1}^{t+1} hM + a(S)$, and the variance of C_{2t+1} is equal to $Q + B^2(2A - a(S)) \geq Q$. Now suppose that $a(S) > A$, which implies that $a(S) \geq A+1$. Hence, the expected value of C_{2t+1} plus the standard deviation of C_{2t+1} is at least equal to

$$\sum_{h=1}^{t+1} hM + A + 1 + \sqrt{Q} > \sum_{h=1}^{t+1} hM + A + 1 + \sqrt{Q + B^2 A} - 1 = d_{2t+1},$$

where the '>' sign comes from the choice of Q. But then J_{2t+1} is late. This contradiction shows that $a(S) = A$, and we have shown that the answer to PARTITION is 'yes' as well.

Since the decision variant of our problem clearly belongs to $\mathcal{NP}$, we have proven the following theorem.

Theorem 4.1 *The problem of minimizing the number of late jobs on a single machine with normally distributed processing times is $\mathcal{NP}$-hard.* ■

Until so far, we have not worried about the possibility that the actual processing time of a job might become negative. This can virtually be avoided in the above reduction by adding to each processing time a huge constant and updating the due dates in a corresponding fashion.

5 Maximizing the expected number of on time jobs

In Section 3 we redefined the concept of on time to be able to establish the value of the objective function without waiting until the last job was finished. Another way to circumvent this problem is to use as an objective function the expected number of on time jobs. Note that in this case we no longer can guarantee a lower bound on the probability that a job is on time. The quality of a sequence σ is then measured as

$$\sum_{j=1}^{n} P(C_j \leq d_j),$$

which expression has to be maximized. Obviously, if not all jobs are equally important, then we can add weights w_j and maximize the total weighted probability. Since a job in a given sequence can be classified as on time or late no longer, we will speak of jobs that are 'accepted' and 'rejected'; we are looking for a schedule in which first the set of accepted jobs are executed in order of due date after which the rejected jobs follow (if they are executed at all). Hence, we need to partition the job set into a set of accepted and a set of rejected jobs. We suppose that a rejected job has zero probability of meeting its due date.

Note that it may seem optimal to execute the accepted jobs in EDD-order, but this is clearly not always true. Consider the following two-job example. Both jobs have a processing time that follows a normal distribution. The data are $p_1 = 12$, $p_2 = 2$, $\sigma_1^2 = 16$, and $\sigma_2^2 = 1$. The due dates are $d_1 = 14$ and $d_2 = 15$. The expected number of on time jobs is equal to almost 1.29 for the EDD-sequence and 1.5 for the reverse sequence. This situation will occur only if the due dates are close to each other, if the completion time of the second job is close to the due date, and if one job is much bigger than the other. As a precaution, we break ties in the EDD-order such that smaller jobs get lower index, but our dynamic programming algorithms developed below are only guaranteed to produce optimal schedules subject to the additional constraint that the accepted jobs are executed in EDD-order.

We start with the instance class with processing times that follow a gamma distribution; the class of jobs with processing times from a negative binomial distribution can be dealt with in the same way. Given a sequence in which J_j is an accepted job, we use S to denote the set of accepted jobs up to and including J_j. To compute $P(C_j \leq d_j)$ we only need $p(S)$. To measure the quality of this solution, we need to know $\sum_{i \in S} P(C_i \leq d_i)$, which we denote as $Pr(S)$. We can capture this information by using statevariables $f_j(t)$ $(j = 0, \ldots, n; t = 0, \ldots, \sum_{h=1}^{j} p_h)$, which denote the maximum value $Pr(S)$ over all subsets S of $\{J_1, \ldots, J_j\}$ with $p(S) = t$. Working this out is straightforward and therefore we do not provide a formal description. A similar algorithm can be used for the instance class with equally disturbed processing times, but now we need to keep track of $|S|$ as well. We therefore use statevariables $f_j(k,t)$ $(j=0,\ldots,n; k=0,\ldots,j; t=0,\ldots,\sum_{h=1}^{j} p_h)$, which again denote the maximum value $Pr(S)$ over all subsets S of $\{J_1,\ldots,J_j\}$ with $|S|=k$ and $p(S)=t$. Finally, the instance class with normally distributed processing times can be dealt with by a similar dynamic programming algorithm. We need to keep track of $p(S)$, $\sigma^2(S)$, and $Pr(S)$ in each iteration. Therefore, we use statevariables $f_j(s,t)$ $(j=0,\ldots,n; s=0,\ldots,\sum_{h=1}^{j}\sigma_h^2; t=0,\ldots,\sum_{h=1}^{j} p_h)$, which denote the maximum value $Pr(S)$ over all subsets S of $\{J_1,\ldots,J_j\}$ with $\sigma^2(S)=s$ and $p(S)=t$.

The weighted versions can be dealt with through straightforward generalizations of the above algorithms. But now the additional constraint of putting the jobs in EDD-order has a larger impact, since it is more attractive to advance a very important job such that it certainly will finish on time and after that process a much less important job with a smaller due date.

6 Concluding remarks

In this paper, we have addressed the single-machine scheduling problem of maximizing the number of jobs that can be completed on time in case of stochastic processing times, where we introduced the concept of minimum success probability to deal with the complications of stochastic processing times. We have considered four probability distributions to draw the processing times from. The gamma and negative binomial distribution can be dealt with by the deterministic algorithm of Moore and Hodgson, even if each job has its own specific minimum success probability. The other two, equal random disturbance and normal distribution, are much more difficult, but can be solved by dynamic programming in case the optimal way to feed the jobs into the dynamic programming algorithm is known. We further have discussed the problems that we face in case of arbitrary minimum success probabilities or in case of maximizing the expected number of on time jobs.

There are several open questions and directions for future research. The first one is to get a tighter characterization of an optimal order to add the jobs in the dynamic programming algorithm. Another direction is to introduce a 'look-forward' time, during which information can be gathered concerning the actual processing times of the first jobs in the schedule. Closely related to this is to put the problem in a dynamic context.

Acknowledgement. The authors want to express their gratitude to Sem Borst for his helpful comments on an earlier draft of the paper.

References

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

[2] R.M. Karp (1972). Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher (eds.) (1972). *Complexity of Computer Computations*, Plenum Press, New York, pp. 85-103.

[3] A.M. Law and W.D. Kelton (2000). *Simulation modeling and analysis.* McGraw Hill.

[4] E.L. Lawler and J.M. Moore (1969). A functional equation and its application to resource allocation and sequencing problems. *Management Science 16*, 77-84.

[5] E.L. Lawler (1973). Optimal sequencing of a single machine subject to precedence constraints. *Management Science 19*, 544-546.

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