Minimizing the number of late jobs in case of stochastic processing times with minimum success probabilities

Marjan van den Akker

Han Hoogeveen

institute of information and computing sciences, utrecht university

technical report UU-CS-2004-067

www.cs.uu.nl

Minimizing the number of late jobs in case of stochastic processing times with minimum success probabilities

Marjan van den Akker * Han Hoogeveen †

Abstract

We consider the single-machine scheduling problem of minimizing the number of late jobs. We omit here one of the standard assumptions in scheduling theory, which is that the processing times are deterministic. Our main message is that in a number of cases the problem with stochastic processing times can be reformulated as a deterministic problem, which is solvable in polynomial time through the famous algorithm by Moore and Hodgson. We first review and reinterpret this algorithm as a dynamic programming algorithm. We then consider four problem classes with stochastic processing times. The first one has equally disturbed processing times, that is, the processing time 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 shape parameter p_j and scale parameter β, where β is common to all jobs; (ii) A negative binomial distribution with parameters p_j and r, where r is the same for each job; (iii) A normal distribution with parameters p_j and σ_j^2.

In this scheduling environment, the completion times will be stochastic variables as well. Instead of looking at the expected number of on time jobs, we introduce the concept of a job being 'stochastically on time', that is, 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 in case of equally disturbed processing times we can solve the problem in $O(n \log n)$ time through the algorithm by Moore and Hodgson, if we make the additional assumption that the due dates and the minimum success probabilities are agreeable, which encompasses the case of equal minimum success probabilities. The problems with processing times following a gamma or a negative binomial distribution can be solved in $O(n \log n)$ time by Moore and Hodgson's algorithm, even if the minimum success probabilities are arbitrary; based on these two examples, we characterize the properties that a distribution must possess to allow such a result. For the case with normally distributed processing times we need the additional assumption that the due dates and minimum success probabilities are agreeable. Under this assumption 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

*E-mail: marjan@cs.uu.nl. Department of Computer Science, Utrecht University, P.O.Box 80089, 3508 TB Utrecht, The Netherlands. Supported by EC Contract IST-1999-14186 (Project alcom-FT).

†E-mail: slam@cs.uu.nl. Department of Computer Science, Utrecht University, P.O.Box 80089, 3508 TB Utrecht, The Netherlands. Supported by EC Contract IST-1999-14186 (Project alcom-FT).

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 a given order.

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 many 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 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 for this objective it does not matter at which time a late job is finished, such a job can just as well be skipped altogether; the machine then only carries out the jobs that will finish on time. Since it is not acceptable for a customer to just hear 'sorry, we did not make it' at the due date, we assume that the firm has to tell the potential client at time zero whether it will honor its request. If the request is denied, then the client will go elsewhere, and the company does not have to execute this job. Note that, if all information is available at time zero in a deterministic situation, then this assumption is in fact irrelevant. 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 common assumptions of scheduling theory, which is that the data are deterministic. 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 p_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 r and malfunction with probability $(1 - r)$; a job J_j corresponds then to an order of p_j correctly functioning items. The corresponding processing time then follows a negative binomial distribution with parameters p_j and r. In the third class the processing times consist

of a deterministic component p_j 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/her order, if the company can convince him/her that the planning is such that the probability that the order is delayed is 'small enough'. This is achieved by guaranteeing that 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*, and which we denote by y_j $(j = 1, \ldots, n)$. If the customer prefers to be convinced by hard cash, then you can agree that he/her will be compensated if the completion is delayed; when the probability distribution of the completion time of job J_j is known, then working with a minimum success probability boils down to specifying an upper bound for the expected compensation payment, which corresponds to a lower bound on the expected profit, and vice versa.

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. We developed this approach ourselves and only found out recently that it has been described in an unpublished paper by Lawler (-) for the more complicated case with *agreeable* release dates r_j $(j = 1, \ldots, n)$, where 'agreeable' in this setting implies that the jobs can be numbered such that $r_i \leq r_j$ implies that $d_i \leq d_j$, for all $i, j = 1, \ldots, n$ (see Van den Akker and Hoogeveen (2004)). 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. We further specify a number of constraints such that, if a probability distribution satisfies these, then the problem with stochastic processing times following this distribution is solvable in $O(n \log n)$ time irrespective of the minimum success probabilities; the first two classes of instances satisfy these conditions. For the other two classes of processing times, we need the additional assumption that the minimum success probabilities and the due dates are *agreeable*, which here implies that the jobs can be numbered such that $i < j$ implies that $d_i \leq d_j$ and $y_i \geq y_j$. We develop dynamic programming algorithms that minimize the (weighted) number of late jobs for these instances; these algorithms are based on 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 any given 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 will use 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 subsets of $\{J_1, \ldots, J_j\}$ with cardinality k that we have to care about are the ones 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 a 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

state-variable $f_j(k)$ 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 z_j and $f_j(k)$ for a given j and all $k = 0, \ldots, z_j$. We first determine z_{j+1} by checking whether job J_{j+1} can be on time given that it succeeds the jobs in $E_j^*(z_j)$. Hence, we get the recurrence relation

$$z_{j+1} = \begin{cases} z_j + 1 & \text{if } f_j(z_j) + p_{j+1} \leq d_{j+1} \\ z_j & \text{otherwise} \end{cases} \tag{1}$$

We put $f_{j+1}(0) = 0$ and determine $f_{j+1}(k)$ $(k = 1, \ldots, z_{j+1})$ through the recurrence relation

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

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 initialization, 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 state-variables $f_j(k)$ 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 *A 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 the optimality of $E_j^*(k)$, 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. Observe that the choice of J_q implies that from J_q onwards the EDD-schedule of $\bar{E}_j^*(k)$ consists of exactly the same jobs as the EDD-schedule of $E_j^*(k)$, but they are started $p(E_j^*(k)) - p(\bar{E}_j^*(k)) > 0$ time units earlier now. 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 $f_j(z_j)$ is equal to $p(E_j^*(z_j))$ and we can compute $f_j(z_j - 1)$ as the total processing time of $E_j^*(z_j)$ minus the processing time of the longest job in $E_j^*(z_j)$. Hence, we know from the recurrence relation (2) that we can compute $E_{j+1}^*(z_{j+1})$ in the following way. First we determine whether $z_{j+1} = z_j$ or $z_{j+1} = z_j + 1$. The criterion is whether $f_j(z_j) + p_{j+1} = p(E_j^*(z_j)) + p_{j+1} \leq d_{j+1}$; if this is the case, then $z_{j+1} = z_j + 1$, and we need J_{j+1} in the early set as an additional job. Otherwise, we have to choose when determining $E_j^*(z_j)$ between ignoring job J_{j+1}, which yields a set with total processing time $p(E_j^*(z_j))$, and replacing the longest job in $E_j^*(z_j)$ with J_{j+1}. Since we want to minimize the total processing time of the early set, we prefer the first option if p_{j+1} is equal to or larger than the largest processing time in the set $E_j^*(z_j)$; otherwise, we prefer the second option. Hence, the choice between these options boils down to finding the largest job, say J_k, in the set $E_j^*(z_j) \cup \{J_{j+1}\}$ and setting $E_{j+1}^*(z_{j+1})$ equal to $E_j^*(z_j) \cup \{J_{j+1}\} \setminus \{J_k\}$. If $J_k = J_{j+1}$, then $E_{j+1}^*(z_{j+1}) = E_j^*(z_j)$, which is feasible; if $J_k \neq J_{j+1}$, then the completion time of J_{j+1} is equal to $p(E_{j+1}^*(z_{j+1})) \leq p(E_j^*(z_j)) \leq d_j \leq d_{j+1}$, from which feasibility follows immediately. Summarizing, we get the following:

- 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.

Hence, the algorithm by Moore and Hodgson can be regarded as a dynamic programming algorithm that works with state-variables $g_j(z_j)$ only, where the state-variable $g_j(z_j)$ then contains

the *subset of the jobs that lead to* $f_j(z_j)$. Since the number of state-variables is $O(n)$, it can be implemented to run in $O(n \log n)$ time, whereas our dynamic programming algorithm requires $O(n^2)$ time.

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 customers beforehand. 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 the order of the jobs; when running the schedule we determine the starting and completion times on line through the 'wait-and-see' approach: as soon as a job is finished, we start the next one in the sequence. This implies that we only know the completion times of the jobs, and hence whether a job was on time or tardy, *after the schedule has been executed*. Due to our assumption that we have to make the decision of accepting or rejecting a job at time zero, we cannot even await the actual processing times of the first couple of jobs in the schedule and use this information in our decision. Hence, we can only be sure that a job will be completed on time if its own processing time and the processing times of all its accepted predecessors satisfy some given upper bound (which exists only if the distribution is bounded), but working with this upper bound as the actual processing time will lead to a very small number of accepted jobs. To overcome this problem, we change our definition of on time as follows, such that it becomes more practicable for the case of stochastic processing times.

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. We call this version of on time* stochastically on time.

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 shape parameter p_j and scale parameter 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 and r, which is equal for all jobs. The negative binomial distribution can be used to model the process of picking p_j correctly functioning items, where each item has a probability of r to function well. The expected number of items that have to be produced such that exactly p_j ones are okay is then equal to p_j/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 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. 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. We further indicate the properties that the distribution of the processing times must possess such that the corresponding problem can be solved in $O(n \log n)$ time irrespective of the minimum success probabilities.

Given an order of the jobs, the completion time of some job J_j is equal to the total processing time of J_j and all its predecessors; from now on, we use Q_j to denote this set of jobs (which includes J_j). Since the processing times are independent random variables following a gamma distribution, we know that $C_j = \sum_{i \in Q_j} \pi_i$ follows a gamma distribution as well, with parameters $\sum_{i \in Q_j} p_i$ and β. Hence, we can compute $P[C_j \leq d_j]$, if we know the value of $p(Q_j) = \sum_{i \in Q_j} p_i$, since β and d_j are known parameters; note that no knowledge of the set Q_j is required except for $p(Q_j)$. 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. Since $P[C_j \leq d_j]$ is a monotone function in the value $p(Q_j)$, we can easily compute D_j as the largest integral value of $p(Q_j)$ such that $P[C_j \leq d_j] \geq y_j$ if C_j follows a gamma distribution with parameters $p(Q_j)$ and β, for instance by applying binary search. This yields the following equivalence.

Theorem 3.2 *Let $\mathcal{I}$ be any instance of the problem of maximizing the number of stochastically on time jobs, where each job J_j ($j = 1, \ldots, n$) has processing time π_j ($j = 1, \ldots, n$), which is gamma distributed with shape parameter p_j and scale parameter β, due date d_j, and minimum success probability y_j. Let $\mathcal{I}'$ be the instance of the deterministic problem of maximizing the number of on time jobs, where each job J_j ($j = 1, \ldots, n$) has processing time p_j and due date D_j, which is computed as described above. Then both problems are solved by the same set of on time jobs.*

Proof. The theorem is proved by showing that any set of on time jobs that is feasible for one problem is also feasible for the other problem. But this is trivial, since $p(Q_j) \leq D_j$ implies $P[C_j \leq d_j] \geq y_j$, given that C_j follows a gamma distribution with parameters $p(Q_j)$ and β, and vice versa. ∎

Hence, we have converted our stochastic problem into an equivalent deterministic problem,

which can be solved to optimality by applying Moore-Hodgson's algorithm in $O(n \log n)$ time. Therefore, we have the following corollary.

Corollary 3.3 *When the processing times are gamma distributed with shape parameter p_j and scale parameter β, the problem of maximizing the number of stochastically on time jobs is solvable in $O(n \log n)$ time.* ■

If we look at the weighted problem, where w_j is the reward of completing J_j on time, 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 (1969) 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 (Karp, 1972), 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.

The above approach worked for the gamma distribution, because of the following two reasons:

- The completion time C_j of job J_j $(j = 1, \ldots, n)$ is fully specified by $p(Q_j)$.
- $P[C_j \leq d_j]$ does not increase when $p(Q_j)$ increases.

It is easily verified that these two conditions are sufficient: if for an instance of the stochastic problem both conditions are satisfied, then the problem can be solved in $O(n \log n)$ time. The first condition is satisfied if the probability distribution is described by only one job-specific parameter and if the probability distribution is additive, that is, if π_i and π_j follow this distribution with specific parameters p_i and p_j, then $\pi_i + \pi_j$ follows this distribution with specific parameter $p_i + p_j$. Hence, we come to the following characterization of an 'easy' class of instances for this problem, which we state without proof.

Theorem 3.4 *A class of instances of the problem of maximizing the number of stochastically on time jobs is solvable in $O(n \log n)$ time if the following conditions are satisfied:*

- *The processing times are independently distributed stochastic variables that are described by one job-specific parameter p_j.*
- *The probability distribution is additive in the job-specific parameter.*
- *$P[C_j \leq d_j] \geq y_j$ does not increase when $p(Q_j)$ increases, irrespective of y_j.* ■

It is easily verified that the class of negative binomial distributed processing times satisfies the conditions of Theorem 3.4, if the processing times are independent. Hence, the unweighted problem is solved in $O(n \log n)$ time, 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.

A similar approach can be used when we do not work with minimum success probabilities but with some kind of reward function per job: a lower bound on the expected profit then leads to an upper bound D_j on $p(Q_j)$, after which the resulting problem of deciding which tasks to carry out can be solved by the algorithm of Moore-Hodgson with deterministic processing times p_j and due dates D_j.

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 Q_j denote the set containing all jobs that have been accepted up to and including J_j. Hence, we have that $C_j = \sum_{i \in Q_j} \pi_i$, follows a distribution that is described by $p(Q_j)$ and the joint disturbance of the jobs in Q_j. This implies, however, that the second condition of Theorem 3.4 is not satisfied, as the number of jobs in Q_j, which we denote by $|Q_j|$, is crucial. 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. We will show in Theorem 3.5 below that adding the jobs in EDD-order, where ties are settled according to non-increasing y_j value, will permit us to find an optimal solution if the minimum success probabilities are agreeable with the due dates, with which we mean that the jobs can be numbered such that $i < j$ implies that $d_i \leq d_j$ and $y_i \geq y_j$, for all $i, j = 1, \ldots, n$. We assume from now on that the jobs have been renumbered in this way; with we a little abuse of notation, we call this order the EDD-order again.

Since all disturbances are independently, identically distributed, the composite disturbance only depends on the number of jobs that belong to Q_j. 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 $p(Q_j)$ and $|Q_j|$. Obviously, if $|Q_j|$ is given, then $P[C_j \leq d_j]$ is maximized by choosing for $Q_j \setminus \{J_j\}$ the subset of $\{J_1, \ldots, J_{j-1}\}$ of minimum total deterministic processing time from among the sets with cardinality $|Q_j| - 1$ that is feasible; in this respect a subset is feasible if the corresponding jobs can be scheduled such that each job is stochastically on time. Therefore, the required knowledge to compute $P[C_j \leq d_j]$ is captured by the information stored in the state-variables $f_j(k)$ 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 checking of whether J_{j+1} is on time has to be modified from inspecting '$f_j(k) + p_{j+1} \leq d_{j+1}$' to inspecting whether '$P[C_{j+1} \leq d_{j+1}] \geq y_{j+1}$', that is, we have to check whether '$P[C_j + \pi_{j+1} \leq d_{j+1}] \geq y_{j+1}$'. What is left to prove is the validity of the EDD-order.

Theorem 3.5 *If $d_i \leq d_j$ and $y_i \geq y_j$, 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 subsequence of the jobs corresponding to any optimal solution. 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 one, since our algorithm is guaranteed to find the best solution corresponding to the order in which the jobs are fed into the algorithm. Hence, it suffices to show that there exists an optimum subsequence 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, as the same subset of jobs is involved, and J_i was on time. Since $d_i \leq d_j$ and $y_i \geq y_j$, 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. ∎

It is straightforward to solve the weighted case by using state-variables $f_j(k, W)$, where W contains the additional information of the total weight of the current set of on time jobs. Note that, in contrast to the deterministic case in Section 2, we now need both the total weight of the current set of on time jobs and the cardinality of this set, since knowledge of the cardinality is required 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 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, where the variance is so small that the probability of getting a negative processing time is negligible. We further assume that all π_j variables are independent of each other. Since the variances can differ per job, this problem is fundamentally more difficult than the previous ones, as we will show in Section 4. Again, we define C_j as the stochastic completion time of J_j $(j = 1, \ldots, n)$, and we let Q_j denote the set containing all jobs that have been selected to complete on time up to and including J_j. Hence, we have that $C_j = \sum_{i \in Q_j} \pi_i$ follows the normal distribution with mean $\sum_{i \in Q_j} p_i$ and variance $\sum_{i \in Q_j} \sigma_i^2$. 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 y_j. Again, we assume that the due dates and minimum success probabilities are agreeable; Theorem 3.5 shows that there exists an optimal schedule that starts with the on time jobs in EDD-order.

Obviously, not all possible combinations $(\sum_{i \in Q_j} p_i, \sum_{i \in Q_j} \sigma_i^2)$ are of interest. If we look at all feasible subsets Q_j of $\{J_1, \ldots, J_j\}$ with equal $\sum_{i \in Q_j} \sigma_i^2$ value and equal cardinality, then our best chance to meet y_j is offered by the subset with minimum $\sum_{i \in Q_j} p_i$ value. On the other hand, if we can choose among all feasible subsets Q_j of $\{J_1, \ldots, J_j\}$ with equal $\sum_{i \in Q_j} p_i$ value and equal cardinality, then we would like to have the one with smallest $\sum_{i \in Q_j} \sigma_i^2$ value if $y_j > 0.5$ and the one with maximum $\sum_{i \in Q_j} \sigma_i^2$ value if $y_j < 0.5$; if $y_j = 0.5$, then the variances are irrelevant, which implies that the problem in which all minimum success probabilities are equal to 0.5 can be solved by applying the algorithm by Moore-Hodgson to the instance with deterministic processing times p_j. Since we may expect that a customer is not satisfied with a probability of less than 50% that the manufacturer completes his job on time, we assume that $y_j \geq 0.5$ for all $j = 1, \ldots, n$. Therefore, 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 Q_j with $p(Q_j) = t$ that has minimum total variance, if it exists. The reason behind enumerating the total processing time and minimizing 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, then we can reverse the role of the variance and the total processing time in the dynamic programming algorithm.

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 state-variable $f_j(k,t)$, which is supposed to denote the minimum variance $\sum_{i \in Q_j} \sigma_i^2$ over all feasible subsets Q_j of $\{J_1, \ldots, J_j\}$ with $|Q_j| = k$ and $\sum_{i \in Q_j} p_i = t$. Here z_j is defined as the maximum cardinality over all feasible subsets Q_j 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 (where ties are settled according to non-increasing y_j value). When adding job J_{j+1}, we first determine z_{j+1}. Thereto, we check for each $f_j(z_j, t)$ whether

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

where z denotes a standard normal variable. If this holds for none of the state-variables $f_j(z_j, t)$, then we put $z_{j+1} = z_j$; otherwise, we put $z_{j+1} = z_j + 1$. As the check above indicates, we have to be a bit cautious. The case that $z_{j+1} = z_j + 1$ does not imply that adding J_{j+1} behind each feasible solution represented by $f_j(z_j, t)$ results in J_{j+1} being stochastically on time. If we encounter a situation in which there is no feasible solution corresponding to $f_j(k,t)$, then we give it the value INF (from 'infeasible'). Given the correct values for $f_j(k', t')$, we compute the values $f_{j+1}(k,t)$ as follows. We put $f_{j+1}(0,0) = 0$ and compute $f_{j+1}(k,t)$ for $k = 1, \ldots, z_{j+1}$ and $t = 0, \ldots, \sum_{i=1}^{j+1} p_i$ 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. We work with the special value INF in the following way. If an entry has value INF, then this possibility is ignored by the minimand; if both entries have value INF, then $f_{j+1}(k,t)$ gets the value INF. Furthermore, if the minimum comes from the second term and the first term has value INF, then we check whether J_{j+1} is stochastically on time when C_{j+1} has expected value t and variance $f_j(k-1, t-p_{j+1}) + \sigma_{j+1}^2$; if this is not the case, then we set $f_{j+1}(k,t)$ equal to INF after all. Note that, if the minimum comes from the second term and the first term is not equal to INF, then we know that J_{j+1} will be stochastically on time, since J_j was stochastically on time in a comparable but less favorable situation. 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 instead of the cardinality of the on time set. Hence, the algorithm then runs in $O(n \sum p_i \sum w_i)$ time.

3.4 The non-agreeable case

Suppose that the due dates and the minimum success probabilities are not agreeable. 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 stochastically 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.

Therefore we now address the question: 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 3.6 *If $d_i \leq d_j$ and $y_i \geq y_j$, where at least one of the inequalities is strict, then we can limit ourselves to orders in which J_i precedes J_j.*

Proof. The proof follows immediately from the proof of Theorem 3.5. ∎

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 with equal minimum success probabilities 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 Q of the index set $\{1, \ldots, t\}$ such that

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

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 Z 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 Z is chosen such that $\sqrt{Z + B^2A} - \sqrt{Z} < 1$; a straightforward computation shows that $Z = A^6(A+1)^2$ 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	Z	$\sum_{h=1}^{t+1} hM + A + \sqrt{Z + B^2A}$

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 not stochastically on time?

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

Now we come to the formal proof. First suppose that the answer to PARTITION is 'yes'; let Q denote the subset of $\{1, \ldots, t\}$ with $\sum_{j \in Q} a_j = A$. 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 $j \notin Q$ and J_{2j} if $j \in Q$, 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^2A, for $i = 1, \ldots, t$, whereas $C_{[t+1]}$ has variance $Z + B^2A$. Note that the minimum success probability is such that 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. Hence, all jobs in E are 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. We prove this statement by showing that any set $\bar{E}$ containing t jobs from $J_1, \ldots, J_{2t}$ for which the above does not hold is not feasible. To facilitate the proof, we alter the instance by reducing the processing times and getting rid of the variances; obviously, if $\bar{E}$ is not feasible for this instance, then $\bar{E}$ is also infeasible with respect to the original instance. We put $p_{2i} = iM$ and $\sigma^2_{2i-1} = 0$ for $i = 1, \ldots, t$; the other data remain unchanged. If we apply Moore-Hodgson's algorithm to this instance, then we find that at most i ($i = 1, \ldots, t$) out 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 $\sum_{h=1}^{i} hM$, for $i = 1, \ldots, t$. Now we take a closer look at the set $\bar{E}$; we define the sets $\bar{E}_j$ ($j = 1, \ldots, t$) as $\bar{E} \cap \{J_1, \ldots, J_{2j}\}$. We first show that $\bar{E}_t$ must contain exactly one job from $\{J_{2t-1}, J_{2t}\}$. Since $|\bar{E}_t| = t$ and 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 $\bar{E}_t$. If $\bar{E}_t$ contains both, then $\bar{E}_t$ contains $t-2$ jobs from $\{J_1, \ldots, J_{2t-2}\}$, and applying Moore-Hodgson shows that the minimum total processing time of such a set is at least $\sum_{h=1}^{t-2} hM + 2tM$. But then

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

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

This implies that the set E corresponding to the 'yes' solution must contain exactly one of each pair of jobs $\{J_{2i-1}, J_{2i}\}$ ($i = 1, \ldots, t$); hence, J_{2t+1} must then also be on time to get the count right. Now that we have established this characteristic of E, we return to our original instance. We define Q 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 $\sum_{h=1}^{t} hM + \sum_{j \in Q} a_j$, and the variance of $C_{[t]}$ is equal to $B^2 \sum_{j \notin Q} a_j$. If $\sum_{j \in Q} a_j < A$, 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 $\sum_{j \in Q} a_j \geq A$.

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

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

where the '$>$' sign comes from the choice of Z. But then J_{2t+1} is late. This contradiction shows that $a(Q) = A$, 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, even if all minimum success probabilities are equal.* ■

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 $\mathcal{NP}$-hardness proof 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 introduced the concept of stochastically 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. The quality of a sequence 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.

The disadvantage, from our point of view, of working with this kind of objective function, is that a job will always be included in the optimal order as long as there is a positive probability of meeting the due date. Therefore, we use the minimum success probabilities in this setting, too, and we only choose jobs that are stochastically on time. Hence, we are now looking for a sequence in which each job is stochastically on time and by which the total probability of meeting the due date is maximized. We solve this problem by applying dynamic programming again. Since we consider a job only once in our dynamic programming algorithm, a job that is rejected, although being stochastically on time, does not get a second chance. This approach is correct only if adding this job to the final set of selected jobs cannot lead to a feasible subset, which can be verified by the analogon of Lawler's *Least-cost-last-rule* (Lawler, 1973) formulated in Subsection 3.4.

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 $d_2 = 15$, and we choose $y_1 = y_2 = 0.5$. The expected number of on time jobs is equal to almost 1.29 for the EDD-sequence and 1.5 for the reverse sequence. Note that, if we would have $y_1 = y_2 = 0.55$, then the reverse sequence would be infeasible, and the EDD-sequence would be optimal. This implies that we can only guarantee that an optimum is found by feeding the jobs to our dynamic programming algorithms in all non-dominated orders. Unfortunately, the above example also shows that a job with larger due date and smaller minimum success probability should sometimes precede a job with a smaller due date and a larger minimum success probability, which makes a partial description of the set of optimal orders unlikely. Given that the jobs are added in order of index, where the initial numbering is arbitrary, our algorithms below find the best solution *in which the accepted jobs are executed in order of index.*

We start with the instance class with processing times that follow a gamma distribution; the class of instances 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 Q_j to denote the set of accepted jobs up to and including J_j. To compute $P(C_j \leq d_j)$ we only need $p(Q_j)$. To measure the quality of this solution, we need to know $\sum_{i \in Q_j} P(C_i \leq d_i)$, which we denote as $Pr(Q_j)$. We can capture this information by using state-variables $f_j(t)$ $(j = 0, \ldots, n; t = 0, \ldots, \sum_{h=1}^{j} p_h)$, which denote the maximum value $Pr(Q_j)$ over all subsets Q_j of $\{J_1, \ldots, J_j\}$ with $p(Q_j) = t$. Working this out is standard, and therefore we omit it for reasons of brevity.

A similar algorithm can be used for the instance class with equally disturbed processing times, but now we need to keep track of $|Q_j|$ as well. We therefore use state-variables $f_j(k, t)$ $(j = 0, \ldots, n; k = 0, \ldots, j; t = 0, \ldots, \sum_{h=1}^{j} p_h)$, which again denote the maximum value $Pr(Q_j)$ over all subsets Q_j of $\{J_1, \ldots, J_j\}$ with $|Q_j| = k$ and $p(Q_j) = t$.

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 $p(Q_j)$, $\sigma^2(Q_j)$, and $Pr(Q_j)$ in each iteration. Therefore, we use state-variables $f_j(s, t)$ $(j = 0, \ldots, n; s = 0, \ldots, \sum_{h=1}^{j} \sigma_h^2; t = 0, \ldots, \sum_{h=1}^{j} p_h)$, which denote the maximum value $Pr(Q_j)$ over all subsets Q_j of $\{J_1, \ldots, J_j\}$ with $\sigma^2(Q_j) = s$ and $p(Q_j) = t$.

The weighted versions can be dealt with through straightforward generalizations of the above algorithms.

6 Concluding remarks

In this paper, we have addressed the accept/reject problem, where an accepted order represents an amount of work, the exact value of which is unknown but that can be modelled through a stochastic variable. When accepted, an order must be ready at a given time, which is called the due date. Since the completion times are stochastic variables, it is hard to guarantee beforehand that this due date will be met. Therefore, we have introduced the concept of the minimum success probability, which gives to some degree a guarantee that an accepted order will be ready on time. This concept is more general than the requirement that the expected completion time is no more than the due date, and it offers the possibility to the client to ask for a higher reliability of on time completion. Moreover, a similar approach can be used in the situation that a penalty has to be paid for tardy completion, which may depend on the actual completion time, and the manufacturer accepts a job only if the expected profit is large enough.

The deterministic problem can be solved by applying dynamic programming algorithms for both the weighted and unweighted case. The question is whether this knowledge can be used in the case of stochastic processing times. We have answered this question for four probability distributions on the processing times; these reflect models that are frequently encountered in practice. The four probability distributions are the gamma distribution, the negative binomial distribution, the normal distribution, and a general case with the characteristic that all processing times are equally disturbed. In all cases we can express the completion times as random variables with an accessible distribution. More importantly, we can transform the question 'is the probability that the job is completed on time at least equal to the minimum success probability?' into the question 'is the sum of the parameters of the accepted jobs no more than some upper bound', which is straightforward deterministic. Hence, we are able to

remove the stochastic features of the problem, such that we can solve it by using the dynamic programming algorithms that have been developed for the deterministic situations. This works especially well for the gamma and negative distribution: we can solve these problems by the deterministic algorithm of Moore and Hodgson, even if each job has its own specific minimum success probability. We have further given a characterization that a probability distribution must possess such that this approach can be used. The other two, equal random disturbance and normal distribution, are much more difficult. Here, we need to know the order in which to feed the jobs into the dynamic programming algorithm. We have identified conditions that partly characterize this order; the problem is solvable if the due dates and minimum success probabilities are agreeable. 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, where we work with a lower bound on the probability of being on time.

There are several open questions and directions for future research. The first one is to derive more dominance rules that must be satisfied by 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 an on line context.

We further want to point out that, whereas we have considered the single machine problem only, a similar approach can be used to solve parallel machine problems with stochastic processing times.

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.M. van den Akker and J.A. Hoogeveen (2004). Minimizing the number of tardy jobs. In: J.Y.-T. Leung (ed.), *Handbook of Scheduling*, Algorithms, Models, and Performance Analysis, pp. 227–243, CRC Press, Inc. Boca Raton, Fl, USA.

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

[3] 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.

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

[5] E.L. Lawler (-). Scheduling a single machine to minimize the number of late jobs. Unpublished manuscript.

[6] 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.

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

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