Contents

14

Stochastic scheduling models

Michael L. Pinedo
New York University

14.1. Preliminaries

Production environments in real life are subject to many sources of uncertainty. These sources of uncertainty include machine breakdowns, unexpected releases of high priority jobs, i.e., jobs with large weights, and the unpredictability of processing times which are often not known in advance. Thus a good model of a scheduling problem would need to address these forms of uncertainty.

The goal of this chapter is not to give an exhaustive overview of the field of stochastic scheduling. It is rather meant to give an overview of stochastic counterparts of the specific deterministic models considered in the previous chapters and draw some comparisons between deterministic models and stochastic models. Because of space limitations, we present at times the known results not in their full generality, especially when an elaborate framework is needed to present such results.

The first section of this chapter goes over the notation, the classes of distributions, the various forms of stochastic dominance, and the different classes of scheduling policies.

In what follows, it is assumed that the *distributions* of the processing times, release dates and due dates are all known in advance, that is, at time zero. The actual *outcome* or *realization* of a random processing time only becomes known upon the completion of the processing; the realization of a release date or due date becomes known only at the point in time at which it actually occurs.

For this chapter we adopt the following notation. Random variables are capitalized, while the actual realized values are in lower case. Job j has the following

quantities of interest associated with it.

X_{ij} = the random processing time of job j on machine i ; if job j is only to be processed on one machine, or if it has the same processing times on each of the machines it may visit, the subscript i is omitted.

$1/\lambda_{ij}$ = the mean or expected value of the random variable X_{ij}.

R_j = the random release date of job j.

D_j = the random due date of job j.

w_j = the weight (or importance factor) of job j.

This notation is not completely analogous to the notation used for the earlier deterministic models. The reason X_{ij} is used as the processing time in stochastic scheduling is due to the fact that P usually refers to a probability. The weight w_j, similar to that in the deterministic models, is basically equivalent to the cost of keeping job j in the system for one unit of time. In the queueing theory literature, which is closely related to stochastic scheduling, c_j is often used for the weight or cost of job j. The c_j and the w_j are equivalent.

Distributions and density functions may take many forms. In what follows, for obvious reasons, only distributions of nonnegative random variables are considered.

A random variable from a continuous time distribution may assume any real nonnegative value within one or more intervals. A distribution function is typically denoted by $F(t)$ and its density function by $f(t)$, i.e.,

$$F(t) = P(X \leq t) = \int_0^t f(t)\,dt,$$

where

$$f(t) = \frac{dF(t)}{dt}$$

provided the derivative exists. Furthermore,

$$\bar{F}(t) = 1 - F(t) = P(X \geq t).$$

An important example of a continuous time distribution is the *exponential* distribution. The density function of an exponentially distributed random variable X is

$$f(t) = \lambda e^{-\lambda t},$$

and the corresponding distribution function is

$$F(t) = 1 - e^{-\lambda t},$$

which is equal to the probability that X is smaller than t. The mean or expected value

of X is

$$E(X) = \int_0^\infty t f(t)\, dt = \int_0^\infty t\, dF(t) = \frac{1}{\lambda}.$$

The parameter λ is called the *rate* of the exponential distribution.

Another important distribution is the *deterministic* distribution. A deterministic random variable assumes a given value with probability one.

The *completion rate* $c(t)$ of a continuous time random variable X with density function $f(t)$ and distribution function $F(t)$ is defined as follows:

$$c(t) = \frac{f(t)}{1 - F(t)}.$$

This completion rate is equivalent to the failure rate or hazard rate in reliability theory. For an exponentially distributed random variable $c(t) = \lambda$ for all t. That the completion rate is independent of t is one of the reasons why the exponential distribution plays an important role in stochastic scheduling. This property is closely related to the *memoryless* property of the exponential distribution, which implies that the distribution of the *remaining* processing time of a job which already has been processed for an amount of time t, is still exponentially distributed with rate λ and therefore identical to its processing time distribution at the very start of its processing.

Distributions can be classified based on their completion rate. An *Increasing Completion Rate (ICR)* distribution is defined as a distribution whose completion rate $c(t)$ is increasing in t, while a *Decreasing Completion Rate (DCR)* distribution is defined as a distribution whose completion rate is decreasing in t.

A subclass of the class of *ICR* distributions is the class of *Erlang(k, λ)* distributions. The *Erlang(k, λ)* distribution is defined as

$$F(t) = 1 - \sum_{r=0}^{k-1} \frac{(\lambda t)^r e^{-\lambda t}}{r!}.$$

The *Erlang(k, λ)* is a k-fold convolution of the exponential distribution and has a mean k/λ. Thus, if k equals one, the distribution is an exponential with mean $1/\lambda$ and if k and λ both go to ∞, then the distribution becomes a constant, i.e., a deterministic distribution.

A subclass of the class of *DCR* distributions is the class of *mixtures of exponentials*. A random variable X is distributed according to a mixture of exponentials if it is exponentially distributed with rate λ_j with probability p_j, $j = 1, \ldots, n$, and

$$\sum_{j=1}^{n} p_j = 1.$$

The exponential as well as the deterministic distribution are special cases of *ICR* distributions. The exponential distribution is *DCR* as well as *ICR*. The class of *DCR*

distributions contains other special distributions. For example, let X be distributed as follows: with probability p it is exponentially distributed with rate p and with probability $1-p$ it is zero. Clearly, $E(X)=1$. When p is very close to zero this distribution will be referred to as an *Extreme Mixture of Exponentials (EME)*.

One way of measuring the variability of a distribution is through its coefficient of variation C_v. The coefficient of variation C_v is defined as the variance of the distribution divided by the square of the mean, i.e.,

$$C_v = \frac{Var(X)}{(E(X))^2} = \frac{E(X^2)-(E(X))^2}{(E(X))^2}.$$

It can be verified easily that the C_v of the deterministic distribution is zero and of the exponential distribution one. The C_v of an extreme mixture of exponentials may be arbitrarily large (it goes to ∞ when p goes to 0).

14.2. Stochastic dominance and classes of policies

In stochastic scheduling random variables often have to be compared with one another. There are many ways in which comparisons between random variables can be made. Comparisons based on certain properties are typically referred to as *stochastic dominance,* i.e., a random variable *dominates* another with respect to some stochastic property.

Definition 14.1. *(i) The random variable X_1 is said to be larger in expectation than the random variable X_2 if $E(X_1) \geq E(X_2)$.*

(ii) The random variable X_1 is said to be stochastically larger than the random variable X_2 if

$$P(X_1 > t) \geq P(X_2 > t)$$

or

$$1 - F_1(t) \geq 1 - F_2(t)$$

for all t. This ordering is usually referred to as stochastic ordering and is denoted by $X_1 \geq_{st} X_2$.

(iii) The random variable X_1 is almost surely larger than or equal to the random variable X_2 if $P(X_1 \geq X_2) = 1$. This ordering implies that the density functions f_1 and f_2 may overlap on at most one point. This ordering is denoted by $X_1 \geq_{a.s.} X_2$.

Ordering in expectation is the crudest form of stochastic dominance. Stochastic ordering implies ordering in expectation since

$$E(X_1) = \int_0^\infty t f_1(t)dt = \int_0^\infty (1-F_1(t))dt = \int_0^\infty \bar{F}_1(t)dt.$$

The three forms of stochastic dominance described above all imply that the random

variables being compared, in general, have different means. They lead to the following chain of implications.

$$\boxed{\text{almost surely larger} \implies \text{stochastically larger} \implies \text{larger in expectation}}$$

There are several other important forms of stochastic dominance that are based on the *variability* of the random variables assuming that the *means are equal.* In the subsequent definitions three such forms are presented. One of these is defined for density functions which are symmetric around the mean, i.e.,

$$f(E(X)+t) = f(E(X)-t)$$

for all $0 \leq t \leq E(X)$. Such a density function then has an upper bound of $2E(X)$.

Definition 14.2. *(i) The random variable X_1 is said to be larger than the random variable X_2 in the variance sense if the variance of X_1 is larger than the variance of X_2.*

(ii) The random variable X_1 is said to be more variable than the random variable X_2 if

$$\int_0^\infty h(t)dF_1(t) \geq \int_0^\infty h(t)dF_2(t)$$

for all convex functions h. This ordering is denoted by $X_1 \geq_{cx} X_2$.

(iii) The random variable X_1 is said to be symmetrically more variable than the random variable X_2 if the density functions $f_1(t)$ and $f_2(t)$ are symmetric around the same mean $1/\lambda$ and $F_1(t) \geq F_2(t)$ for $0 \leq t \leq 1/\lambda$ and $F_1(t) \leq F_2(t)$ for $1/\lambda \leq t \leq 2/\lambda$.

Again, the first form of stochastic dominance is somewhat crude. However, any two random variables with equal means can be compared with one another in this way.

From the fact that the functions $h(t) = t$ and $h(t) = -t$ are convex, it follows that if X_1 is "more variable" than X_2 then $E(X_1) \geq E(X_2)$ and $E(X_1) \leq E(X_2)$. So $E(X_1)$ has to be equal to $E(X_2)$. From the fact that $h(t) = t^2$ is convex it follows that $Var(X_1)$ is larger than $Var(X_2)$. Variability ordering is a partial ordering, i.e., not every pair of random variables with equal means can be ordered in this way. At times, variability ordering is also referred to as ordering *in the convex sense*.

It can be shown easily that symmetrically more variable implies more variable in the convex sense but not vice versa.

The forms of stochastic dominance described in *Definition 15.2* lead to the following chain of implications:

$$\boxed{\text{symmetrically more variable} \implies \text{more variable} \implies \text{larger in variance}}$$

In stochastic scheduling, certain conventions have to be made which are not

needed in deterministic scheduling. During the evolution of a stochastic process new information becomes available continuously. Job completions and occurrences of random release dates and due dates provide additional information that the decision-maker may wish to take into account while scheduling the remaining jobs. The amount of freedom the decision maker has in using this additional information is the basis for the various classes of decision making policies. In this section four classes of policies are defined.

The first class of policies is, in what follows, only used in scenarios where all the jobs are available for processing at time zero; the machine environments considered are the single machine, parallel machines and permutation flow shops.

Definition 14.3. *Under a nonpreemptive static list policy the decision maker orders the jobs at time zero according to a priority list. This priority list does not change during the evolution of the process and every time a machine is freed the next job on the list is selected for processing.*

Under this class of policies the decision maker puts at time zero the n jobs in a list (permutation) and the list does not change during the evolution of the process. In the case of machines in parallel, every time a machine is freed, the job at the head of the list is selected as the next one for processing. In the case of a permutation flow shop the jobs are also put in a list in front of the first machine at time zero; every time the first machine is freed the next job on the list is scheduled for processing. This class of nonpreemptive static list policies is in what follows also referred to as the class of *permutation* policies. This class of policies is in a sense similar to the static priority rules often considered in deterministic models.

Example 14.4. *Consider a single machine and three jobs. All three jobs are available at time zero. All three jobs have the same processing time distributions, which is 2 with probability .5 and 8 with probability .5. The due date distributions are the same, too. The due date is 1 with probability .5 and 5 with probability .5. If a job is completed at the same time as its due date, it is considered to be on time. It would be of interest to know the expected number of jobs completed in time under a permutation policy.*

Under a permutation policy the first job is completed in time with probability .25 (its processing time has to be 2 and its due date has to be 5); the second job is completed in time with probability .125 (the processing times of the first and second job have to be 2 and the due date of the second job has to be 5); the third job never will be completed in time. The expected number of on-time completions is therefore .375 and the expected number of tardy jobs is $3 - 0.375 = 2.625$.

The second class of policies is a preemptive version of the first class and is in what follows only used in scenarios where jobs are released at *different* points in time.

Definition 14.5. *Under a preemptive static list policy the decision maker orders the jobs at time zero according to a priority list. This ordering includes jobs with*

nonzero release dates, i.e., jobs which are to be released later. This priority list does not change during the evolution of the process and at any point in time the job at the top of the list of available jobs is the one to be processed on the machine.

Under this class of policies the following may occur. When there is a job release at some time point and the job released is higher on the static list than the job currently being processed, then the job being processed is preempted and the job released is put on the machine.

Under the third and fourth class of policies, the decision-maker is allowed to make his decisions during the evolution of the process. That is, every time he makes a decision, he may take all information available at that time into account. The third class of policies does not allow preemptions.

Definition 14.6. *Under a nonpreemptive dynamic policy, every time a machine is freed, the decision maker is allowed to determine which job goes next. His decision at such a time point may depend on all the information available, e.g., the current time, the jobs waiting for processing, the jobs currently being processed on other machines and the amount of processing these jobs already have received on these machines. However, the decision maker is not allowed to preempt; once a job begins processing, it has to be completed without interruption.*

Example 14.7. *Consider the same problem as in* Example 15.4. *It is of interest to know the expected number of jobs completed in time under a nonpreemptive dynamic policy. Under a nonpreemptive dynamic policy the probability the first job is completed in time is again .25. With probability .5 the first job is completed at time 2. With probability .25 the due dates of both remaining jobs already occurred at time 1 and there will be no more on-time completions. With probability .75 at least one of the remaining two jobs has a due date at time 5. The probability that the second job put on the machine is completed in time is 3/16 (the probability that the first job has completion time 2 times the probability at least one of the two remaining jobs has due date 5 times the probability that the second job has processing time 2). Again, it is impossible to start the third job and complete it in time. The expected number of on-time completions is therefore .4375 and the expected number of tardy jobs is 2.5625.*

The last class of policies is a preemptive version of the third class.

Definition 14.8. *Under a preemptive dynamic policy, at any point in time, the decision maker is allowed to select the jobs to be processed on the machines. His decision at a time point may depend on all information available and may require preemptions.*

Example 14.9. *Consider again the problem of* Example 15.4. *It is of interest to know the expected number of jobs completed in time under a preeemptive dynamic policy. Under a preemptive dynamic policy, the probability that the first job is completed*

in time is again .25. This first job is either taken off the machine at time 1 (with probability .5) or at time 2 (with probability .5). The probability the second job put on the machine is completed in time is 3/8, since the second job enters the machine either at time 1 or at time 2 and the probability of being completed on time is 0.75 times the probability it has processing time 2, which equals 3/8 (regardless of when the first job was taken off the machine). However, unlike under the nonpreeemptive dynamic policy, the second job put on the machine is taken off with probability .5 at time 3 and with probability 0.5 at time 4. So now there is actually a chance that the third job that goes on the machine will be completed in time. The probability the third job is completed in time is 1/16 (the probability that the due date of the first job is 1 (=.5) times the probability that the due dates of both remaining jobs are 5 (=.25) times the probability that the processing time of the third job is 2 (=.5)). The total expected number of on-time completions is therefore $11/16 = 0.6875$ *and the expected number of tardy jobs is 2.3125.*

It is clear that the optimal preemptive dynamic policy leads to the best possible value of the objective as in this class of policies the decision maker has the most information available and the largest amount of freedom. It is also clear that if all jobs are present at time zero and the environment is either a bank of machines in parallel or a permutation flow shop, then the optimal nonpreemptive dynamic policy is at least as good as the optimal nonpreemptive static list policy (see *Examples 15.4* and *15.7*).

There are several forms of minimization in stochastic scheduling. Whenever an objective function has to be minimized, it should be specified in what *sense* the objective is to be minimized. The crudest form of optimization is in the *expectation* sense, e.g., one wishes for example to minimize the *expected* makespan, that is $E(C_{\max})$ and find a policy under which the expected makespan is smaller than the expected makespan under any other policy. A stronger form of optimization is optimization in the *stochastic* sense. If a schedule or policy minimizes $C_{\max}$ stochastically, the makespan under the optimal schedule or policy is *stochastically* less than the makespan under any other schedule or policy. Stochastic minimization, of course, implies minimization in expectation. In the subsequent sections the objective is usually minimized in expectation. Frequently, however, the policies that minimize the objective in expectation minimize the objective stochastically as well.

14.3. Single machine models

Stochastic models, especially with exponential processing times, may often contain more structure than their deterministic counterparts and lead to results which, at first sight, may seem surprising. Models that are NP-hard in a deterministic setting often allow a simple priority policy to be optimal in a stochastic setting.

In this section we first consider single machine models with arbitrary processing times in a nonpreemptive setting. Then we analyze models with exponentially distributed processing times.

For a number of stochastic problems, finding the optimal policy is equivalent to solving a deterministic scheduling problem. Usually, when such an equivalence relationship exists, the deterministic counterpart can be obtained by replacing all random variables with their means. The optimal schedule for the deterministic problem then minimizes the objective of the stochastic version in expectation.

One such case is when the objective in the deterministic counterpart is linear in $p_{(j)}$ and $w_{(j)}$, where $p_{(j)}$ and $w_{(j)}$ denote the processing time and weight of the job in the jth position in the sequence.

This observation implies that it is easy to find the optimal permutation schedule for the stochastic counterpart of $1 \mid\mid \sum w_j C_j$, when the processing time of job j is X_j, from an arbitrary distribution F_j, and the objective is $E(\sum w_j C_j)$. This problem leads to the stochastic version of the *WSPT* rule, which sequences the jobs in decreasing order of the ratio $w_j/E(X_j)$ or $\lambda_j w_j$. In what follows this rule is referred to as the *Weighted Shortest Expected Processing Time first (WSEPT)* rule or as the "λw" rule.

Theorem 14.10. *The WSEPT rule minimizes the expected sum of the weighted completion times in the class of nonpreemptive static list policies as well as in the class of nonpreemptive dynamic policies.*

Proof. The proof for nonpreemptive static list policies is similar to the proof for the deterministic counterpart of this problem. The proof is based on an adjacent pairwise interchange argument identical to the one used for the deterministic counterpart of this problem. The only difference is that the p_j's in that proof have to be replaced by the $E(X_j)$'s.

The proof for nonpreemptive dynamic policies needs an additional argument. It is easy to show that it is true for $n = 2$ (again an adjacent pairwise interchange argument). Now consider three jobs. It is clear that the last two jobs have to be sequenced according to the λw rule. These last two jobs will be sequenced in this order independent of what happens during the processing of the first job. There are then three sequences that may occur: each of the three jobs starting first and the remaining two jobs sequenced according to the λw rule. A simple interchange argument between the first job and the second shows that all three jobs have to sequenced according to the λw rule. It can be shown by induction that all n jobs have to be sequenced according to the λw rule in the class of nonpreemptive dynamic policies: suppose it is true for $n-1$ jobs. If there are n jobs it follows from the induction hypothesis that the last $n-1$ jobs have to be sequenced according to the λw rule. Suppose the first job is not the job with the highest $\lambda_j w_j$. Interchanging this job with the second job in the sequence, i.e., the job with the highest $\lambda_j w_j$, leads to an decrease in the expected value of the objective function. This completes the proof of the theorem. □

It can be shown that the nonpreemptive *WSEPT* rule is also optimal in the class of *preemptive* dynamic policies when all n processing time distributions are *ICR*. This follows from the fact that any time when a preemption is contemplated, the $w_j/E(X_j)$ ratio of the job currently on the machine is actually higher than it was when first put on the machine (the expected remaining processing time of an *ICR* job decreases as

processing goes on). If the ratio of the job was the highest among the remaining jobs when it was put on the machine, it remains the highest while it is being processed.

The same cannot be said about jobs with *DCR* distributions. The expected remaining processing time then *increases* while a job is being processed. So the weight divided by the expected remaining processing time of a job, while it is being processed, *decreases* with time. Preemptions may be thus advantageous with *DCR* processing times.

Example 14.11. *Consider n jobs with the processing time X_j distributed as follows. The processing time X_j is 0 with probability p_j and it is distributed according to an exponential with rate λ_j with probability $1-p_j$. Clearly, this distribution is* DCR *as it is a mixture of two exponentials with rates ∞ and λ_j. The objective to be minimized is the expected sum of the weighted completion times. The optimal preemptive dynamic policy is clear. All n jobs have to be tried out for a split second at time zero, in order to determine which jobs have zero processing times. If a job does not have zero processing time, it is taken immediately off the machine. The jobs with zero processing times are then all completed at time zero. After determining in this way which jobs have nonzero processing times, these remaining jobs are sequenced in decreasing order of $\lambda_j w_j$.*

The remaining part of this section focuses on due date related problems. Consider the stochastic counterpart of $1 \mid\mid L_{\max}$ with processing times having arbitrary distributions and deterministic due dates. The objective is to minimize the expected maximum lateness.

Theorem 14.12. *The EDD rule minimizes expected maximum lateness for arbitrarily distributed processing times and deterministic due dates in the class of nonpreemptive static list policies, the class of nonpreemptive dynamic policies and the class of preemptive dynamic policies.*

Proof. It is clear that the *EDD* rule minimizes the maximum lateness for any realization of processing times (after conditioning on the processing times, the problem is basically a deterministic problem and the results for the deterministic counterpart of this problem apply). If the *EDD* rule minimizes the maximum lateness for any realization of processing times then it minimizes the maximum lateness also in expectation (it actually minimizes the maximum lateness with probability 1). □

It can be shown that the *EDD* rule not only minimizes

$$E(L_{\max}) = E(\max(L_1, \ldots, L_n)),$$

but also $\max(E(L_1), \ldots, E(L_n))$. It is even possible to develop an algorithm for a stochastic counterpart of the more general $1 \mid prec \mid h_{\max}$ problem. In this problem the objective is to minimize the maximum of the n expected costs incurred by the n

jobs, i.e., the objective is to minimize

$$\max\Big(E(h_1(C_1)),\ldots,E(h_n(C_n))\Big),$$

where $h_j(C_j)$ is the cost incurred by job j being completed at C_j. The cost function h_j is nondecreasing in the completion time C_j. The algorithm is a modified version of the algorithm for the deterministic counterpart of this problem. The version here is also a *backward* procedure. Whenever one has to select a schedulable job for processing, it is clear that the distribution of its completion time is the convolution of the processing times of the jobs that have not yet been scheduled. Let f_{J^c} denote the density function of the convolution of the processing times of the set of unscheduled jobs J^c. Job j^* is then selected to be processed last among the set of jobs J^c if

$$\int_0^\infty h_{j^*}(t) f_{J^c}(t)dt = \min_{j\in J^c}\int_0^\infty h_j(t) f_{J^c}(t)dt.$$

The *L.H.S.* denotes the expected value of the penalty for job j^* if it is the last job to be scheduled among the jobs in J^c. This rule replaces one step in the algorithm for the deterministic counterpart of this problem. The proof of optimality is similar to the proof of optimality in the deterministic case. However, implementation of the algorithm is significantly more cumbersome as the evaluation of the integrals may not be easy.

We now discuss due date models with exponentially distributed processing times. Consider the stochastic version of $1 \mid d_j = d \mid \sum w_j U_j$ with job j having an exponentially distributed processing time with rate λ_j and a deterministic due date d. Recall that the deterministic counterpart is equivalent to the *knapsack* problem. The objective to be minimized is the expected weighted number of tardy jobs.

Theorem 14.13. *The WSEPT rule minimizes the expected weighted number of tardy jobs in the classes of nonpreemptive static list policies, nonpreemptive dynamic policies and preemptive dynamic policies.*

Proof. First the optimality of the *WSEPT* rule in the class of nonpreemptive static list policies is shown. Assume the machine is free at some time t and two jobs, with weights w_1 and w_2 and processing times X_1 and X_2, remain to be processed. Consider first the sequence $1,2$. The probability that both jobs are late is equal to the probability that X_1 is larger than $d-t$, which is equal to $\exp(-\lambda(d-t))$. The penalty for being late is then equal to w_1+w_2. The probability that only the second job is late corresponds to the event where the processing time of the first job is $x_1 < d-t$ and the sum of the processing times $x_1+x_2 > d-t$. Evaluation of the probability of this event, through conditioning on X_1 (that is $X_1 = x$), yields

$$P(X_1 < d-t, X_1+X_2 > d-t) = \int_0^{d-t} e^{-\lambda_2(d-t-x)}\lambda_1 e^{-\lambda_1 x}dx.$$

If $E(\sum wU(1,2))$ denotes the expected value of the penalty due to jobs 1 and 2, with

job 1 processed first, then

$$E\Big(\sum wU(1,2)\Big) = (w_1+w_2)e^{-\lambda_1(d-t)} + w_2\int_0^{d-t} e^{-\lambda_2(d-t-x)}\lambda_1 e^{-\lambda_1 x}dx.$$

The value of the objective function under sequence $2,1$ can be obtained by interchanging the subscripts in the expression above. Straightforward computation yields

$$E\Big(\sum wU(1,2)\Big) - E\Big(\sum wU(2,1)\Big) =$$

$$(\lambda_2 w_2 - \lambda_1 w_1)\frac{e^{-\lambda_1(d-t)} - e^{-\lambda_2(d-t)}}{\lambda_2-\lambda_1}.$$

It immediately follows that the difference in the expected values is positive if and only if $\lambda_2 w_2 > \lambda_1 w_1$. Since this result holds for all values of d and t, any permutation schedule that does not sequence the jobs in decreasing order of $\lambda_j w_j$ can be improved by swapping two adjacent jobs, where the first has a lower λw value than the second. This completes the proof of optimality for the class of nonpreemptive static list policies.

Induction can be used to show optimality in the class of nonpreemptive dynamic policies. It is immediate that this is true for 2 jobs (it follows from the same pairwise interchange argument for optimality in the class of nonpreemptive static list policies). Assume that it is true for $n-1$ jobs. In the case of n jobs this implies that the scheduler after the completion of the first job will, because of the induction hypothesis, revert to the *WSEPT* rule among the remaining $n-1$ jobs. It remains to be shown that the scheduler has to select the job with the highest $\lambda_j w_j$ as the first one to be processed. Suppose the decision-maker selects a job which does not have the highest $\lambda_j w_j$. Then, the job with the highest value of $\lambda_j w_j$ is processed second. Changing the sequence of the first two jobs decreases the expected value of the objective function according to the pairwise interchange argument used for the nonpreemptive static list policies.

To show that *WSEPT* is optimal in the class of preemptive dynamic policies, suppose a preemption is contemplated at some point in time. The remaining processing time of the job then on the machine is exponentially distributed with the same rate as it had at the start of its processing (because of the memoryless property of the exponential). Since the decision to put this job on the machine did not depend on the value of t at that moment or on the value of d, the same decision remains optimal at the moment a preemption is contemplated. A nonpreemptive policy is therefore optimal in the class of preemptive dynamic policies. □

This result is in marked contrast with the result for its deterministic counterpart, i.e., the knapsack problem, which is NP-hard.

The *WSEPT* rule does not necessarily yield an optimal schedule when processing time distributions are not all exponential.

Theorem 14.13 can be generalized to consider breakdown and repair. Suppose the machine goes through "uptimes", when it is functioning and "downtimes" when it is being repaired. This breakdown and repair may form an arbitrary stochastic process. *Theorem 14.13* also holds under these more general conditions since no part of the proof depends on the remaining time till the due date.

Theorem 14.13 can also be generalized to include different release dates with arbitrary distributions. Assume a finite number of releases after time 0, say n^*. It is clear from the results presented above that at the time of the last release the *WSEPT* policy is optimal. This may actually imply that the last release causes a preemption (if, at that point in time, the job released is the job with the highest $\lambda_j w_j$ ratio in the system). Consider now the time-epoch of the second last release. After this release a preemptive version of the *WSEPT* rule is optimal. To see this, disregard for a moment the very last release. All the jobs in the system at the time of the second to last release (*not* including the last release) have to be sequenced according to *WSEPT*; the last release may in a sense be considered a random "downtime". From the previous results it follows that all the jobs in the system at the time of the second last release should be scheduled according to preemptive *WSEPT*, independent of the time period during which the last release is processed. Proceeding inductively towards time zero it can be shown that a preemptive version of *WSEPT* is optimal with arbitrarily distributed releases in the classes of preemptive static list policies and preemptive dynamic policies.

The *WSEPT* rule also turns out to be optimal for other objectives as well. Consider the stochastic counterpart of $1 \mid d_j = d \mid \sum w_j T_j$ with job j again exponentially distributed with rate λ_j. All n jobs are released at time 0. The objective is to minimize the sum of the expected weighted tardinesses.

Theorem 14.14. *The WSEPT rule minimizes the expected sum of the weighted tardinesses in the classes of nonpreemptive static list policies, nonpreemptive dynamic policies and preemptive dynamic policies.*

Proof. The objective $w_j T_j$ can be approximated by a sum of an infinite sequence of $w_j U_j$ unit penalty functions, i.e.,

$$w_j T_j = \sum_{l=0}^{\infty} w_j U_{jl}.$$

The first unit penalty U_{j0} corresponds to a due date d, the second unit penalty U_{j1} corresponds to a due date $d+\varepsilon$, the third corresponds to a due date $d+2\varepsilon$ and so on. From *Theorem 14.13* it follows that λw rule minimizes each one of these unit penalty functions. If the rule minimizes each one of these unit penalty functions, it also minimizes their sum. □

This theorem can be generalized along the lines of *Theorem 14.13* to include arbitrary breakdown and repair processes and arbitrary release processes, provided all jobs have due date d (including those released after d).

Actually, a generalization in a slightly different direction is also possible. Consider the stochastic counterpart of the problem $1 \mid\mid \sum w_j h(C_j)$. In this model the jobs have no specific due dates, but are all subject to the *same* cost function h. The objective is to minimize $E(\sum w_j h(C_j))$. Clearly, $\sum w_j h(C_j)$ is a simple generalization of $\sum w_j T_j$ when all jobs have the same due date d. The function h can again be approximated by a sum of an infinite sequence of unit penalties, the only difference being that the due dates of the unit penalties are not necessarily equidistant as in the proof of *Theorem 14.14.*

Consider now a stochastic counterpart of the problem $1 \mid\mid \sum w_j h_j(C_j)$, with each job having a *different* cost function. Again, all jobs are released at time 0. The objective is to minimize the total expected cost. The following ordering among cost functions is of interest: a cost function h_j is said to be *steeper* than a cost function h_k if

$$\frac{dh_j(t)}{dt} \geq \frac{dh_k(t)}{dt}$$

for every t, provided the derivatives exist. This ordering is denoted by $h_j \geq_s h_k$. If the functions are not differentiable for every t, the steepness ordering requires

$$h_j(t+\delta) - h_j(t) \geq h_k(t+\delta) - h_k(t),$$

for every t and δ. Note that a cost function being steeper than another does not necessarily imply that it is higher.

Theorem 14.15. *If $\lambda_j w_j \geq \lambda_k w_k \Longleftrightarrow h_j \geq_s h_k$, then the WSEPT rule minimizes the total expected cost in the classes of nonpreemptive static list policies, nonpreemptive dynamic policies and preemptive dynamic policies.*

Proof. The proof follows from the fact that any increasing cost function can be approximated through the proper addition of a (possibly infinite) number of unit penalties at different due dates. If two cost functions, which may be at different levels, go up in the same way over an interval $[t_1, t_2]$, then a series of identical unit penalties go into effect within that interval for both jobs. It follows from *Theorem 14.13* that the jobs have to be sequenced in decreasing order of λw in order to minimize the total expected penalties due to these unit penalties. If one cost function is steeper than another in a particular interval, then the steeper cost function has one or more unit penalties going into effect within this interval, which the other cost function has not. To minimize the total expected cost due to these unit penalties, the jobs have to be sequenced again in decreasing order of λw. □

The results in this section indicate that scheduling problems with exponentially distributed processing times allow for more elegant structural results than their deterministic counterparts. The deterministic counterparts of most of the models discussed in this section are NP-hard. It is intuitively acceptable that a deterministic problem may be NP-hard while its counterpart with exponentially distributed processing times allows for a very simple policy to be optimal. The reason is the fol-

lowing: all data being deterministic (that is, perfect data) makes it very hard for the scheduler to optimize. In order to take advantage of all the information available the scheduler has to spend an inordinately long time doing the optimization. On the other hand when the processing times are stochastic, the data are fuzzier. The scheduler, with less data at hand, will spend less time performing the optimization. The fuzzier the data, the more likely a simple priority rule minimizes the objective in expectation. Expectation is akin to optimizing for the average case.

14.4. Parallel machine models

This section deals with parallel machine models that are stochastic counterparts of the models discussed in Chapter 7–9. The body of knowledge in the stochastic case is considerably less extensive than in the deterministic case.

The results focus mainly on the expected makespan, the total expected completion time and the expected number of tardy jobs. In what follows the number of machines is usually limited to two. Some of the proofs can be extended to more than two machines, but such extensions usually require more elaborate notation. Since these extensions would not provide any additional insight, they are not presented here. The proofs for some of the structural properties of the stochastic models tend to be more involved than the proofs for the corresponding properties of their deterministic counterparts.

The first part of this section deals with nonpreemptive models; the results in this part are obtained through interchange techniques. The second part focuses on preemptive models; the results in this part are obtained through dynamic programming approaches. The third part deals with due date related models.

The first part of this section considers optimal policies in the class of nonpreemptive static list policies and in the class of nonpreemptive dynamic policies. Since preemptions are not allowed, the main technique for determining optimal policies is based on pairwise interchanges. The exponential distribution is considered in detail as its special properties makes the analysis relatively easy.

Consider *two* machines in parallel and n jobs. The processing time of job j is equal to the random variable X_j, that is exponentially distributed with rate λ_j. The objective is to minimize $E(C_{\max})$. Note that this problem is a stochastic counterpart of $P2 \mid\mid C_{\max}$, which is known to be NP-hard. However, in *Section 15.3* it already became clear that scheduling environments with exponentially distributed processing times often have structural properties which their deterministic counterparts do not have. It turns out that this is also the case with machines in parallel.

A nonpreemptive static list policy is followed. The jobs are put into a list and at time zero the two jobs at the top of the list begin processing on the two machines. When a machine becomes free the next job on the list is put on the machine. It is not specified in advance on which machine each job will be processed, nor is it known a priori which job will be the last one to be completed.

Let Z_1 denote the time when the second to last job is completed, i.e., the first

time a machine becomes free with no jobs on the list to replace it. At this time the other machine is still processing its last job. Let Z_2 denote the time that the last job is completed on the other machine (i.e., Z_2 equals the makespan $C_{\max}$). Let the difference D be equal to $Z_2 - Z_1$. It is clear that the random variable D depends on the schedule. It is easy to see that minimizing $E(D)$ is equivalent to minimizing $E(C_{\max})$. This follows from

$$Z_1 + Z_2 = 2C_{\max} - D = \sum_{j=1}^{n} X_j,$$

which is a constant independent of the schedule.

In what follows, a slightly more general two-machine problem is considered for reasons that will become clear later. It is assumed that one of the machines is not available at time zero and becomes available only after a random time X_0, distributed exponentially with rate λ_0. The random variable X_0 may be thought of as the processing time of an additional job which takes precedence and *must* go first. Let $D(X_0, X_1, X_2, \ldots, X_n)$ denote the random variable D, under the assumption that, at time zero, a job with remaining processing time X_0 is being processed on one machine and a job with processing time X_1 is being started on the other. When one of the two machines is freed a job with processing time X_2 is started, and so on. The next lemma, which we present without proof, examines the effect on D of changing a schedule by swapping consecutive jobs 1 and 2.

Lemma 14.16. *For any* λ_0 *and for* $\lambda_1 = \min(\lambda_1, \lambda_2, \ldots, \lambda_n)$

$$E(D(X_0, X_1, X_2, \ldots, X_n)) \leq E(D(X_0, X_2, X_1, \ldots, X_n)).$$

This lemma constitutes a crucial element in the proof of the following theorem.

Theorem 14.17. *The LEPT rule minimizes the expected makespan in the class of nonpreemptive static list policies when there are two machines in parallel and exponentially distributed processing times.*

Proof. By contradiction. Suppose that a different rule is optimal. Suppose that according to this presumed optimal rule, the job with the longest expected processing time is not scheduled for processing either as the first or the second job. (Note that the first and second job are interchangeable as they both start at time zero). Then an improvement can be obtained by performing a pairwise interchange between this longest job and the job immediately preceding this job in the schedule, as by *Lemma 14.16* this reduces the expected difference between the completion times of the last two jobs. Through a series of interchanges it can be shown that the longest job has to be one of the first two jobs in the schedule. In the same way it can be shown that the second longest job has to be among the first two jobs as well. The third longest job can be moved into the third position to improve the objective, and so on. With each

interchange the expected difference, and thus the expected makespan, are reduced. □

The approach used in proving the theorem is basically an adjacent pairwise interchange argument. However, this pairwise interchange argument is not identical to the pairwise interchange arguments used in single machine scheduling. In pairwise interchange arguments applied to single machine problems, *no* restrictions were made on the relation between the two jobs to be interchanged and those that come after them. In *Lemma 14.16* jobs not involved in the interchange have to satisfy a special condition, viz., one of the two jobs being interchanged must have a larger expected processing time than all jobs following it. Requiring such a condition has certain implications. When no special conditions are required, an adjacent pairwise interchange argument actually yields two results: it shows that one schedule minimizes the objective while the *reverse* schedule maximizes that same objective. With a special condition like the one in *Lemma 14.16* the argument works only in one direction. It actually can be shown that the *SEPT* rule does *not* always maximize $E(D)$ among nonpreemptive static list policies.

The result presented in *Theorem 14.17* differs from the results obtained for its deterministic counterpart considerably. One difference is the following: minimizing makespan in a deterministic setting requires only an optimal *partition* of the n jobs over the two machines. After the allocation has been determined, the set of jobs allocated to a specific machine may be sequenced in any order. With exponential processing times, a sequence is determined in which the jobs are to be *released* in order to minimize the expected makespan. No deviation is allowed from this sequence and it is not specified at time zero how the jobs will be partitioned between the machines. This depends on the evolution of the process.

In contrast with the results of *Section 15.3,* which do not appear to hold for distributions other than the exponential, the *LEPT* rule does minimize the expected makespan for other distributions as well.

Consider the case where the processing time of job j is distributed according to a mixture of two exponentials, i.e., with probability p_{1j} according to an exponential with rate λ_1 and with probability p_{2j} $(= 1 - p_{1j})$ according to an exponential with rate λ_2. Assume $\lambda_1 < \lambda_2$. So

$$P(X_j > t) = p_{1j}e^{-\lambda_1 t} + p_{2j}e^{-\lambda_2 t}.$$

This distribution can be described as follows: when job j is put on the machine a (biased) coin is tossed. Dependent upon the outcome of the toss the processing time of job j is either exponential with rate λ_1 or exponential with rate λ_2. After the rate has been determined this way the distribution of the remaining processing time of job j does not change while the job is being processed. So each processing time is distributed according to one of the two exponentials with rates λ_1 and λ_2.

The subsequent lemma again examines the effect on D of an interchange between two consecutive jobs 1 and 2 on two machines in parallel. Assume again that X_0

denotes the processing time of a job 0 with an exponential distribution with rate λ_0. This rate λ_0 may be different from either λ_1 or λ_2.

Lemma 14.18. *For arbitrary λ_0, if $p_{11} \geq p_{12}$, i.e., $E(X_1) \geq E(X_2)$, then*

$$E(D(X_0, X_1, X_2, \ldots, X_n)) \leq E(D(X_0, X_2, X_1, \ldots, X_n)).$$

Note that there are no conditions on λ_0 ; the rate λ_0 may or may not be equal to either λ_1 or λ_2. Through this lemma the following theorem can be shown rather easily.

Theorem 14.19. *The LEPT rule minimizes the expected makespan in the class of nonpreemptive static list policies when there are two machines in parallel and when the processing times are distributed according to a mixture of two exponentials with rates λ_1 and λ_2.*

Proof. Any permutation schedule can be transformed into the *LEPT* schedule through a series of adjacent pairwise interchanges between a longer job and a shorter job immediately preceding it. With each interchange $E(D)$ decreases because of *Lemma 14.18.* □

Showing that *LEPT* minimizes the expected makespan can be done in this case without any conditions on the jobs that are not part of the interchange. This is in contrast with *Theorem 14.17,* where the jobs following the jobs in the interchange had to be smaller than the largest job involved in the pairwise interchange. The additional condition requiring the other expected processing times to be smaller than the expected processing time of the larger of the two jobs in the interchange, is *not* required in this case.

Theorem 14.19 can be extended to include mixtures of three exponentials, with rates λ_1, λ_2 and ∞. The next example also shows that the *LEPT* rule does not necessarily minimize the expected makespan.

Example 14.20. *Let p_{1j} denote the probability job j is exponentially distributed with rate λ_1 and p_{2j} the probability it is distributed with rate λ_2. Assume $\lambda_1 < \lambda_2$. The probability that the processing time of job j is zero is $p_{0j} = 1 - p_{1j} - p_{2j}$. Through similar arguments as the ones used in* Lemma 14.18 *and* Theorem 14.19 *it can be shown that in order to minimize the expected makespan the jobs in the optimal nonpreemptive static list policy have to be ordered in decreasing order of p_{1j}/p_{2j}. The jobs with the zero processing times again do not play a role in the schedule. Clearly, the optimal sequence is not necessarily* LEPT.

The following example is a continuation of the previous example and an illustration of the *Largest Variance first (LV)* rule.

Example 14.21. *Consider the special case of the previous example with*

$$\frac{1}{\lambda_1} = 2$$

and

$$\frac{1}{\lambda_2} = 1.$$

Let

$$\begin{aligned} p_{0j} &= a_j \\ p_{1j} &= a_j \\ p_{2j} &= 1 - 2a_j \end{aligned}$$

So

$$E(X_j) = \frac{p_{1j}}{\lambda_1} + \frac{p_{2j}}{\lambda_2} = 1,$$

for all j and

$$Var(X_j) = 1 + 4a_j.$$

From the previous example it follows that sequencing the jobs in decreasing order of p_{1j}/p_{2j} minimizes the expected makespan. This rule is equivalent to scheduling the jobs in decreasing order of $a_j/(1-2a_j)$. As $0 \leq a_j \leq 1/2$, scheduling the jobs in decreasing order of $a_j/(1-2a_j)$ is equivalent to scheduling in decreasing order of a_j, which in turn is equivalent to the Largest Variance first *rule.*

The methodology used in proving that *LEPT* is optimal for the expected makespan on two machines does not easily extend to problems with more than two machines or problems with other processing time distributions. Consider the following generalization of this approach for m machines. Let Z_1 denote the time that the first machine becomes idle with no jobs waiting for processing, Z_2 the time the second machine becomes idle, and so on and let Z_m denote the time the last machine becomes idle. Clearly Z_m equals the makespan. Let

$$D_i = Z_{i+1} - Z_i \qquad i = 1, \ldots, m-1.$$

From the fact that the sum of the processing times is

$$\sum_{j=1}^{n} X_j = \sum_{i=1}^{m} Z_i = mC_{\max} - D_1 - 2D_2 - \cdots - (m-1)D_{m-1},$$

independent of the schedule, it follows that minimizing the makespan is equivalent to minimizing

$$\sum_{i=1}^{m-1} iD_i.$$

A limited number of processing time distributions can be handled this way. For example, the settings of *Theorem 14.19* and *Examples 14.20* and *14.21* can be extended relatively easily through this approach. However, the scenario of *Theorem 14.17* cannot be extended that easily.

So far only the class of nonpreemptive static list policies has been considered in this section. It turns out, that most optimal policies in the class of nonpreemptive static list policies are also optimal in the classes of nonpreemptive dynamic policies and preemptive dynamic policies. The proof that a nonpreemptive static list policy is optimal in these other two classes of policies is based on induction arguments very similar to the ones described in the second and third parts of the proof of *Theorem 14.13.*

In what follows an entirely different approach is presented which first proves optimality in the class of preemptive dynamic policies. As the optimal policy is a nonpreemptive static list policy, the policy is also optimal in the the classes of nonpreemptive dynamic policies and nonpreemptive static list policies.

Only the expected makespan has been considered so far in this section. The total expected completion time $E(\sum C_j)$ in a nonpreemptive setting is a slightly more difficult objective to deal with than the expected makespan. Indeed, an approach similar to the one used to show that *LEPT* minimizes the expected makespan for exponential processing times, has not been found to show that *SEPT* minimizes the total expected completion time. However, if the processing times are distributed as in *Theorem 14.19,* it can be shown that *SEPT* minimizes the expected flow time and if the processing times are distributed as in *Example 14.21* it can be shown that *LV* minimizes the expected flow time.

Pairwise interchange arguments are basically geared to determine optimal policies in the class of nonpreemptive static list policies. After determining an optimal nonpreemptive static list policy it can often be argued that this policy is also optimal in the class of nonpreemptive dynamic policies and possibly in the class of preemptive dynamic policies.

In what follows an alternative proof for *Theorem 14.17* is presented. The approach is entirely different. A dynamic programming type proof is constructed within the class of *preemptive* dynamic policies. After obtaining the result that the nonpreemptive *LEPT* policy minimizes the expected makespan in the class of preemptive dynamic policies, it is concluded that it is also optimal in the class of nonpreemptive dynamic policies as well as in the class of nonpreemptive static list policies.

The approach can be used for proving that *LEPT* minimizes the expected makespan for m machines in parallel. It will be illustrated for 2 machines in parallel since the notation is much simpler.

Suppose $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. Let $V(J)$ denote the expected value of the minimum remaining time needed (that is, under the optimal policy) to finish all jobs given that all the jobs in the set $J = j_1, \ldots, j_l$ already have been completed; if $J = \emptyset$, then $V(J)$ is simply denoted by V. Let $V^*(J)$ denote the same time quantity under the *LEPT* policy. Similarly, V^* denotes the expected value of the remaining completion time under *LEPT* when no job has yet been completed.

Theorem 14.22. *The nonpreemptive LEPT policy minimizes the expected makespan in the class of preemptive dynamic policies.*

Proof. The proof is by induction on the number of jobs. Suppose that the result is true when there are less than n jobs. It has to be shown that it is also true when there are n jobs. That is, a policy which at time 0 (when there are n jobs waiting for processing) does not act according to *LEPT* but at the first job completion (when there are $n-1$ jobs remaining to be processed) switches over to *LEPT* results in a larger expected makespan than when *LEPT* is adopted immediately from time zero on.

Conditioning on the first job completion yields

$$V = \min_{j,k} \Big(\frac{1}{\lambda_j + \lambda_k} + \frac{\lambda_j}{\lambda_j + \lambda_k} V^*(\{j\}) + \frac{\lambda_k}{\lambda_j + \lambda_k} V^*(\{k\}) \Big).$$

The expected time until the first job completion is the first term on the *R.H.S.*; the second (third) term is equal to the probability of job j (k) being the first job to be completed, multiplied by the expected remaining time needed to complete the $n-1$ remaining jobs under *LEPT*. This last equation is equivalent to

$$0 = \min_{j,k} \Big(1 + \lambda_j \Big(V^*(\{j\}) - V^* \Big) + \lambda_k \Big(V^*(\{k\}) - V^* \Big) + (\lambda_j + \lambda_k) \Big(V^* - V \Big) \Big).$$

Since λ_1 and λ_2 are the two smallest λ_j values and supposedly $V^* \geq V$ the fourth term on the *R.H.S.* is minimized by $\{j,k\} = \{1,2\}$. Hence to show that *LEPT* is optimal it suffices to show that $\{j,k\} = \{1,2\}$ also minimizes the sum of the second and third term. In order to simplify the presentation let

$$A_j = \lambda_j \Big(V^*(\{j\}) - V^* \Big)$$

and

$$D_{jk} = A_j - A_k.$$

In order to show that

$$\lambda_j \Big(V^*(\{j\}) - V^* \Big) + \lambda_k \Big(V^*(\{k\}) - V^* \Big) = A_j + A_k$$

is minimized by $\{j,k\} = \{1,2\}$, it suffices to show that $\lambda_j < \lambda_k$ implies $A_j \leq A_k$ or, equivalently, $D_{jk} \leq 0$. To prove that $D_{jk} \leq 0$ is done in what follows by induction.

Throughout the remaining part of the proof V^*, A_j and D_{jk} are considered functions of the random variables $\lambda_1, \ldots, \lambda_n$. Define $A_j(J)$ and $D_{jk}(J)$, assuming jobs j and k are not in J, in the same way as A_j and D_{jk}, e.g.,

$$A_j(J) = \lambda_j \Big(V^*(J \cup \{j\}) - V^*(J) \Big).$$

Before proceeding with the induction a number of identities have to be established. If j and k are the two smallest jobs not in the set J, then jobs j and k will, under *LEPT*, be processed first. Conditioning on the first job completion results in the identity

$$V^*(J) = \frac{1}{\lambda_j+\lambda_k} + \frac{\lambda_j}{\lambda_j+\lambda_k}V^*(J\cup\{j\}) + \frac{\lambda_k}{\lambda_j+\lambda_k}V^*(J\cup\{k\})$$

or

$$(\lambda_j+\lambda_k)V^*(J) = 1+\lambda_j V^*(J\cup\{j\}) + \lambda_k V^*(J\cup\{k\}).$$

Similarly,

$$\begin{aligned}(\lambda_1+\lambda_2+\lambda_3)A_1 &= \lambda_1(\lambda_1+\lambda_2+\lambda_3)V^*(\{1\}) - \lambda_1(\lambda_1+\lambda_2+\lambda_3)V^* \\ &= \lambda_1\Big(1+\lambda_1V^*(\{1\})+\lambda_2V^*(\{1,2\})+\lambda_3V^*(\{1,3\})\Big) \\ &\quad -\lambda_1\Big(1+\lambda_1V^*(\{1\})+\lambda_2V^*(\{2\})+\lambda_3V^*\Big) \\ &= \lambda_1\Big(\lambda_3V^*(\{1,3\})-\lambda_3V^*(\{1\})\Big) \\ &\quad +\lambda_2\Big(\lambda_1V^*(\{1,2\})-\lambda_1V^*(\{2\})\Big)+\lambda_3A_1\end{aligned}$$

or

$$(\lambda_1+\lambda_2)A_1 = \lambda_1A_3(\{1\})+\lambda_2A_1(\{2\}).$$

The following identities can be established in the same way:

$$(\lambda_1+\lambda_2)A_2 = \lambda_1A_2(\{1\})+\lambda_2A_3(\{2\})$$

and

$$(\lambda_1+\lambda_2)A_j = \lambda_1A_j(\{1\})+\lambda_2A_j(\{2\}),$$

for $j=3,\ldots,n$. Thus, with $D_{12}=A_1-A_2$, it follows that

$$D_{12} = \frac{\lambda_1}{\lambda_1+\lambda_2}D_{32}(\{1\}) + \frac{\lambda_2}{\lambda_1+\lambda_2}D_{13}(\{2\}),$$

and

$$D_{2j} = \frac{\lambda_1}{\lambda_1+\lambda_2}D_{2j}(\{1\}) + \frac{\lambda_2}{\lambda_1+\lambda_2}D_{3j}(\{2\}),$$

for $j=3,\ldots,n$.

Assume now as induction hypothesis that if $\lambda_j<\lambda_k$, and $\lambda_1\le\cdots\le\lambda_n$, then

$$D_{jk}\le 0$$

and

$$\frac{dD_{12}}{d\lambda_1}\ge 0.$$

In the remaining part of the proof, these two inequalities are shown by induction on n. When $n = 2$,

$$D_{jk} = \frac{\lambda_j - \lambda_k}{\lambda_j + \lambda_k}$$

and the two inequalities can be established easily.

Assume that the two inequalities of the induction hypothesis hold when there are less than n jobs remaining to be processed. The induction hypothesis now implies that $D_{13}(\{2\})$ as well as $D_{23}(\{1\})$ are nonpositive when there are n jobs remaining to be completed. It also provides

$$\frac{dD_{13}(\{2\})}{d\lambda_1} \geq 0.$$

This last inequality has the following implication: if λ_1 increases then $D_{13}(\{2\})$ increases. The moment λ_1 reaches the value of λ_2 jobs 1 and 2 become interchangeable. Therefore

$$D_{13}(\{2\}) \leq D_{23}(\{1\}) = -D_{32}(\{1\}) \leq 0.$$

From the fact that $\lambda_1 < \lambda_2$ it follows that D_{12} is nonpositive. The induction hypothesis also implies that $D_{2j}(\{1\})$ and $D_{3j}(\{2\})$ are nonpositive, whereby D_{2j} is nonpositive. This completes the induction argument for the first inequality of the induction hypothesis. The induction argument for the second inequality can be established by differentiating

$$\frac{\lambda_1}{\lambda_1 + \lambda_2} D_{32}(\{1\}) + \frac{\lambda_2}{\lambda_1 + \lambda_2} D_{13}(\{2\})$$

with respect to λ_1 and then using induction to show that every term is positive. □

This proof shows that *LEPT* is optimal in the class of preemptive dynamic policies. As the optimal policy is a nonpreemptive static list policy it also has to be optimal in the class of nonpreemptive static list policies as well as in the class of nonpreemptive dynamic policies. In contrast with the first proof of the same result, this approach also works for an arbitrary number of machines in parallel. The notation, however, becomes significantly more involved.

The interchange aproach described in the beginning of this section is not entirely useless. To show that the nonpreemptive *LEPT* policy is optimal when the processing time distributions are *ICR*, one has to adopt a pairwise interchange type argument. The reason is obvious. In a preemptive framework the remaining processing time of an *ICR* job that has received a certain amount of processing may become less (in expectation) than the expected processing time of a job that is waiting for processing. This then would lead to a preemption. The approach used in *Lemma 14.16* and *Theorem 14.17* can be applied easily to a number of different classes of distributions for which the approach used in *Theorem 14.22* does not appear to yield the optimal nonpreemptive schedule.

Consider minimizing the total expected completion time of the n jobs. The pro-

cessing time of job j is exponentially distributed with rate λ_j. In Chapter 7 it was shown that the *SPT* rule is optimal for the deterministic counterpart of this problem. This gives an indication that in a stochastic setting the *Shortest Expected Processing Time first (SEPT)* rule may minimize the sum of the expected completion times under appropriate conditions. Consider again two machines in parallel with n jobs. The processing time of job j is exponentially distributed with rate λ_j. An approach similar to the one followed in *Theorem 14.22* for the makespan can be followed for the total expected completion time. The result then is that the nonpreemptive SEPT policy minimizes the total expected completion time in the class of preemptive dynamic policies.

Actually, it turns out that a much more general result can be shown. Consider the more general setting where the n processing times $X_1, \ldots, X_n$ come from arbitrary distributions $F_1, \ldots, F_n$ and $X_1 \leq_{st} X_2 \leq_{st} \cdots \leq_{st} X_n$.

Theorem 14.23. *The nonpreemptive* SEPT *policy minimizes the total expected completion time in expectation and even stochastically in the class of nonpreemptive dynamic policies.*

This result is more general than the corresponding result obtained for minimizing the expected makespan. Recall that *LEPT* does *not* minimize the expected makespan when the $X_1, \ldots, X_n$ are arbitrarily distributed and stochastically ordered.

In the remaining part of this section we consider the problem of two machines in parallel with *i.i.d.* job processing times distributed exponentially with mean 1, with precedence constraints in the form of an intree, and the expected makespan to be minimized in the class of preemptive dynamic policies (that is, a stochastic counterpart of $P2 \mid p_j = 1, intree \mid C_{\max}$). For the deterministic version of this problem the *Critical Path (CP)* rule (sometimes also referred to as the *Highest Level first (HL)* rule) is optimal. The *CP* rule is in the deterministic case optimal for an arbitrary number of machines in parallel, not just two.

For the stochastic version the following notation is needed. The root of the intree is level 0. A job is at level k if there is a chain of $k-1$ jobs between it and the root of the intree. A precedence graph G_1 with n jobs is said to be *flatter* than a precedence graph G_2 with n jobs if the number of jobs at or below level k in G_1 is larger than the number of jobs at or below level k in graph G_2. This is denoted by $G_1 \prec_{fl} G_2$. In the following lemma two scenarios, both with two machines and n jobs but with different intrees, are compared. Let $E(C_{\max}(i)(CP))$ denote the expected makespan under the *CP* rule when the precedence constraints graph takes the form of intree G_i, $i = 1, 2$.

Note that in the subsequent lemma and theorem, preemptions are allowed. However, it will become clear afterwards that for intree precedence constraints the *CP* rule does not require any preemptions. Also, recall that whenever a job is completed on one machine, the remaining processing time of the job being processed on the other machine is still exponentially distributed with mean one.

Theorem 14.24. *The nonpreemptive CP rule minimizes the expected makespan in the class of nonpreemptive dynamic policies and in the class of preemptive dynamic policies.*

As mentioned before, the results presented in *Theorems 14.17* and *14.22*, even though they were only proved for $m = 2$, hold for arbitrary m. The *CP* rule in *Theorem 14.24* is, however, not necessarily optimal for m larger than two.

Example 14.25. *Consider three machines and 12 jobs. The jobs are all* i.i.d. *exponential with mean 1 and subject to the precedence constraints described in Figure ... Scheduling according to the* CP *rule would put jobs 1, 2 and 3 at time zero on the three machines. However, straightforward algebra shows that starting with jobs 1, 2 and 4 results in a smaller expected makespan.*

In the deterministic setting discussed in Chapter 10 it was shown that the *CP* rule is optimal for $P \mid p_j = 1, intree \mid C_{\max}$ and $P \mid p_j = 1, outtree \mid C_{\max}$. One may expect the *CP* rule to be optimal when all processing times are exponential with mean 1 and precedence constraints take the form of an outtree. However, a counterexample can be found easily already in the case of 2 machines in parallel.

Consider again the problem of two machines in parallel with jobs having *i.i.d.* exponentially distributed processing times and subject to precedence constraints which take the form of an intree, but now with the expected flow time as the objective to be minimized. We present the following theorem without proof.

Theorem 14.26. *The nonpreemptive CP rule minimizes the total expected completion time in the class of nonpreemptive dynamic policies and in the class of preemptive dynamic policies.*

14.5. Stochastic multi-operation models

Results for stochastic flow shop, open shop and job shop models are somewhat limited in comparison with the results for their deterministic counterparts.

For flow shops nonpreemptive static list policies, i.e., permutation schedules, are considered first. The optimal permutation schedules often remain optimal in the class of nonpreemptive dynamic policies as well as in the class of preemptive dynamic policies. For open shops and job shops, only the classes of nonpreemptive dynamic policies and preemptive dynamic policies are considered.

The results obtained for stochastic flow shops and job shops are somewhat similar to those obtained for deterministic flow shops and job shops. Stochastic open shops are, however, very different from their deterministic counterparts.

The first section discusses stochastic flow shops with unlimited intermediate storage and jobs not subject to blocking. The second section deals with stochastic flow shops with zero intermediate storage; the jobs are subject to blocking. The last section goes over stochastic open shops and stochastic job shops.

Consider two machines in series with unlimited storage between the machines and no blocking. There are n jobs. The processing time of job j on machine 1 is X_{1j}, exponentially distributed with rate λ_j. The processing time of job j on machine 2 is X_{2j}, exponentially distributed with rate μ_j. The objective is to find the nonpreemptive static list policy or permutation schedule that minimizes the expected makespan $E(C_{\max})$.

Note that this problem is a stochastic counterpart of the deterministic problem $F2 \mid\mid C_{\max}$. The deterministic two machine problem has a very simple solution, i.e., Johnson's rule. It turns out that the stochastic version with exponential processing times has a very elegant solution as well.

Theorem 14.27. *Sequencing the jobs in decreasing order of $\lambda_j - \mu_j$ minimizes the expected makespan in the class of nonpreemptive static list policies, the class of nonpreemptive dynamic policies and the class of preemptive dynamic policies.*

Proof. The proof of optimality in the class of nonpreemptive static list policies is in a sense similar to the proof of optimality in the deterministic case. It is by contradiction. Suppose another sequence is optimal. Under this sequence, there must be two adjacent jobs, say job j followed by job k, such that $\lambda_j - \mu_j < \lambda_k - \mu_k$. It suffices to show that a pairwise interchange of these two jobs reduces the expected makespan. Assume job l precedes job j and let C_{1l} (C_{2l}) denote the (random) completion time of job l on machine 1 (2). Let $D_l = C_{2l} - C_{1l}$.

Perform an adjacent pairwise interchange on jobs j and k. Let C_{1k} and C_{2k} denote the completion times of job k on the two machines under the original, supposedly optimal, schedule and let C'_{1j} and C'_{2j} denote the completion times of job j under the schedule obtained after the pairwise interchange. Let m denote the job following job k. Clearly, the pairwise interchange does not affect the starting time of job m on machine 1 as this starting time is equal to $C_{1k} = C'_{1j} = C_{1l} + X_{1j} + X_{1k}$. Consider the random variables

$$D_k = C_{2k} - C_{1k}$$

and

$$D'_j = C'_{2j} - C'_{1j}.$$

Clearly, $C_{1k} + D_k$ is the time at which machine 2 becomes available for job m under the original schedule, while $C_{1k} + D'_j$ is the corresponding time after the pairwise interchange. First it is shown that the random variable D'_j is stochastically smaller than the random variable D_k. If $D_l \geq X_{1j} + X_{1k}$, then clearly $D_k = D'_j$. The case $D_l \leq X_{1j} + X_{1k}$ is slightly more complicated. Now

$$P(D_k > t \mid D_l \leq X_{1j} + X_{1k}) = \frac{\mu_j}{\lambda_k + \mu_j} e^{-\mu_k t} + \frac{\lambda_k}{\lambda_k + \mu_j} \left(\frac{\mu_k}{\mu_k - \mu_j} e^{-\mu_j t} - \frac{\mu_j}{\mu_k - \mu_j} e^{-\mu_k t} \right).$$

This expression can be explained as follows. Since $D_l \leq X_{1j} + X_{1k}$, then, whenever job j starts on machine 2, job k is either being started or still being processed on machine 1. The first term on the *R.H.S.* corresponds to the event where job j's

processing time on machine 2 finishes before job k's processing time on machine 1, which happens with probability $\mu_j/(\mu_j+\lambda_k)$. The second term corresponds to the event where job j finishes on machine 2 after job k finishes on machine 1; in this case the distribution of D_k is a convolution of an exponential with rate μ_j and an exponential with rate μ_k.

An expression for $P(D'_j > t \mid D_l \leq X_{1j}+X_{1k})$ can be obtained by interchanging the subscripts j with the subscripts k. Now

$$P(D'_j > t \mid D_l \leq X_{1j}+X_{1k}) - P(D_k > t \mid D_l \leq X_{1j}+X_{1k}) = \frac{\mu_j\mu_k}{(\lambda_j+\mu_k)(\lambda_k+\mu_j)} \frac{e^{-\mu_j t}-e^{-\mu_k t}}{\mu_k-\mu_j}(\lambda_j+\mu_k-\lambda_k-\mu_j) \leq 0.$$

So D'_j is stochastically smaller than D_k. It can be shown easily, through a straightforward sample path analysis (i.e., fixing the processing times of job m and of all the jobs following job m), that if the realization of D'_j is smaller than the realization of D_k, then the actual makespan after the interchange is smaller than or equal to the actual makespan under the original sequence before the interchange. So, given that D'_j is stochastically smaller than D_k, the expected makespan is reduced by the interchange. This completes the proof of optimality in the class of nonpreemptive static list (i.e., permutation) policies.

That the rule is also optimal in the class of nonpreemptive dynamic policies can be argued as follows. It is clear that the sequence on machine 2 does not matter. This is because the time machine 2 remains busy processing available jobs is simply the sum of their processing times and the order in which this happens does not affect the makespan. Consider the decisions which have to be made every time machine 1 is freed. The last decision to be made is at that point in time when there are only two jobs remaining to be processed on machine 1. From the pairwise interchange argument described above, it immediately follows that the job with the highest $\lambda_j - \mu_j$ value has to go first. Suppose that there are three jobs remaining to be processed on machine 1. From the previous argument it follows that the last two of these three have to be processed in decreasing order of $\lambda_j - \mu_j$. If the first one of the three is not the one with the highest $\lambda_j - \mu_j$ value, a pairwise interchange between the first and the second reduces the expected makespan. So the last three jobs have to be sequenced in decreasing order of $\lambda_j - \mu_j$. Continuing in this manner it is shown that sequencing the jobs in decreasing order of $\lambda_j - \mu_j$ is optimal in the class of nonpreemptive dynamic policies.

That the nonpreemptive rule is also optimal in the class of preemptive dynamic policies can be shown in the following manner. It is shown above that in the class of nonpreemptive dynamic policies the optimal rule is to order the jobs in decreasing order of $\lambda_j - \mu_j$. Suppose during the processing of a job on machine 1 a preemption is considered. The situation at this point in time is essentially no different from the situation at the point in time the job was started (because of the memoryless property of the exponential distribution). So, every time a preemption is contemplated, the optimal decision is to keep the current job on the machine. Thus the permutation

policy is also optimal in the class of preemptive dynamic policies. □

From the statement of the theorem, it appears that the number of optimal schedules in the exponential case is often smaller than the number of optimal schedules in the deterministic case. The following example makes this clear.

Example 14.28. *Consider n jobs with exponentially distributed processing times. One job has zero processing time on machine 1 and a processing time on machine 2 with a very large mean. Assume that this mean is larger than the sum of the expected processing times of the remaining* $n-1$ *jobs on machine 1. According to* Theorem 14.27 *these remaining* $n-1$ *jobs still have to be ordered in decreasing order of* $\lambda_j - \mu_j$ *for the sequence to minimize the expected makespan.*

If all the processing times were deterministic with processing times equal to the means of the exponential processing times, it would not have mattered in what order the remaining $n-1$ *jobs were sequenced.*

Although at first glance *Theorem 14.27* does not appear to be very similar to Johnson's result for its deterministic counterpart, the optimal schedule with exponential processing times is somewhat similar to the optimal schedule with deterministic processing times. If job k follows job j in the optimal sequence with exponential processing times, then

$$\lambda_j - \mu_j \geq \lambda_k - \mu_k$$

or

$$\lambda_j + \mu_k \geq \lambda_k + \mu_j$$

or

$$\frac{1}{\lambda_j + \mu_k} \leq \frac{1}{\lambda_k + \mu_j},$$

which, with exponential processing times, is equivalent to

$$E(\min(X_{1j}, X_{2k})) \leq E(\min(X_{1k}, X_{2j})).$$

This adjacency condition is quite similar to the condition for job k to follow job j in a deterministic setting, namely

$$\min(p_{1j}, p_{2k}) \leq \min(p_{1k}, p_{2j}).$$

There is another similarity between exponential and deterministic settings. Consider the case where the processing times of job j on both machines are *i.i.d.* exponentially distributed with the same rate, λ_j, for each j. According to the theorem all sequences must have the same expected makespan. This result is similar to the deterministic proportionate flow shop, where all sequences also result into the same makespan.

We now focus on m-machine *permutation* flow shops. For these flow shops only the class of nonpreemptive static list policies is of interest, since the order of the jobs, once determined, is not allowed to change.

Consider an m-machine permutation flow shop where the processing times of job j on the m machines are *i.i.d.* according to distribution F_j with mean $1/\lambda_j$. For such a flow shop it is easy to obtain a lower bound for $E(C_{\max})$.

Lemma 14.29. *Under any sequence*

$$E(C_{\max}) \geq \sum_{j=1}^{n} \frac{1}{\lambda_j} + (m-1)\max\left(\frac{1}{\lambda_1}, \ldots, \frac{1}{\lambda_n}\right)$$

Proof. The expected time it takes the job with the largest expected processing time to traverse the flow shop is at least $m\max(1/\lambda_1, \ldots, 1/\lambda_n)$. The time that this largest job starts on machine 1 is the sum of the processing times on the first machine of those jobs scheduled before the longest job. After the longest job completes its processing on the last machine, this machine remains busy for a time that is at least as large as the sum of the processing times on the last machine of all those jobs scheduled after the longest job. The lemma thus follows. □

One class of sequences plays an important role in stochastic permutation flow shops. A sequence $j_1, \ldots, j_n$ is called a *SEPT-LEPT* sequence, if there is a job j_k in this sequence such that

$$\frac{1}{\lambda_{j_1}} \leq \frac{1}{\lambda_{j_2}} \leq \cdots \leq \frac{1}{\lambda_{j_k}}$$

and

$$\frac{1}{\lambda_{j_k}} \geq \frac{1}{\lambda_{j_{k+1}}} \geq \cdots \geq \frac{1}{\lambda_{j_n}}.$$

Both the *SEPT* and the *LEPT* sequence are examples of *SEPT-LEPT* sequences.

Theorem 14.30. *If $F_1 \leq_{a.s.} F_2 \leq_{a.s.} \cdots \leq_{a.s.} F_n$, then*
(i) any SEPT-LEPT sequence minimizes the expected makespan in the class of non-preemptive static list policies and

$$E(C_{\max}) = \sum_{j=1}^{n} \frac{1}{\lambda_j} + (m-1)\frac{1}{\lambda_n}.$$

(ii) the SEPT sequence minimizes the expected flow time in the class of nonpreemptive static list policies and

$$E(\sum_{j=1}^{n} C_j) = m\sum_{j=1}^{n} \frac{1}{\lambda_j} + \sum_{j=1}^{n-1} \frac{j}{\lambda_{n-j}}.$$

It is easy to find examples with $F_1 \leq_{a.s.} F_2 \leq_{a.s.} \cdots \leq_{a.s.} F_n$, where sequences which are not *SEPT-LEPT* are also optimal (it can be shown that when $F_1, F_2, \ldots, F_n$ are deterministic *any* sequence minimizes the makespan). However, in contrast with

deterministic proportionate flow shops, when processing times are stochastic and $F_1 \leq_{a.s.} F_2 \leq_{a.s.} \cdots \leq_{a.s.} F_n$ not *all* sequences are always optimal.

Example 14.31. *Consider a flow shop with 2 machines and 3 jobs. Job 1 has a deterministic processing time of 11 time units. Job 2 has a deterministic processing time of 10 time units. The processing time of job 3 is zero with probability 0.5 and 10 with probability 0.5. It can be verified easily that* only SEPT-LEPT *sequences minimize the expected makespan. If the processing time of job 1 is changed from 11 to 20, then all sequences have the same expected makespan.*

Consider a two-machine open shop where the processing time of job j on machine 1 is the random variable X_{1j}, distributed according to F_{1j}, and on machine 2 the random variable X_{2j}, distributed according to F_{2j}. The objective is to minimize the expected makespan. As before, the exponential distribution is considered first. In this case, however, it is not known what the optimal policy is when F_{1j} is exponential with rate λ_j and F_{2j} exponential with rate μ_j. It appears that the optimal policy may not have a simple structure and may even depend on the values of the λ's and μ's. The special case where $\lambda_j = \mu_j$ can be analyzed. In contrast with the results obtained for the stochastic flow shops the optimal policy now cannot be regarded as a permutation sequence, but rather as a policy which prescribes a given action dependent upon the state of the system.

Theorem 14.32. *The following policy minimizes the expected makespan in the class of preemptive dynamic policies as well as in the class of nonpreemptive dynamic policies: whenever a machine is freed, the scheduler selects from the jobs which have not yet undergone processing on either one of the two machines, the job with the largest expected processing time. If there are no such jobs remaining the decision-maker may take any job which only needs processing on the machine just freed. Preemptions never need to take place.*

It appears to be very hard to generalize this result to include a larger class of distributions.

Example 14.33. *Let the processing time of job j on machine i, $i = 1,2$, be a mixture of an exponential with rate λ_j and zero with arbitrary mixing probabilities. The optimal policy is to process at time 0 all jobs for a very short period on both machines just to check whether their processing times on the two machines are zero or positive. After the nature of all the processing times have been determined, the problem is reduced to the scenario covered by* Theorem 14.32.

Theorem 14.32 states that jobs which still have to undergo processing on both machines have priority over jobs which only have to be processed on one machine. In a sense, the policy described in *Theorem 14.32* is similar to the *Longest Alternate Processing Time first (LAPT)* rule for the deterministic $O2 \,||\, C_{\max}$ problem.

From *Theorem 14.32* it follows that the problem is tractable also if the processing time of job j on machine 1 as well as on machine 2 is exponentially distributed with rate 1. The policy that minimizes the expected makespan always gives priority to jobs that have not yet undergone processing on either machine. This particular rule does not require any preemptions. In the literature, it has been referred to in this scenario as the *Longest Expected Remaining Processing Time first (LERPT)* rule.

Actually, if in the two-machine case all processing times are exponential with mean 1 and if preemptions are allowed, then the sum of the expected completion times can also be analyzed. This model is an exponential counterpart of $O2 \mid p_{ij} = 1, pmtn \mid \sum C_j$. The total expected completion time clearly requires a different policy. One particular policy is appealling in the class of preemptive dynamic policies: consider the policy which prescribes the scheduler to process, whenever possible, on each one of the machines a job which already has been processed on the other machine. This policy may require the scheduler at times to interrupt the processing of a job and start with the processing of a job which just has completed its operation on the other machine. In what follows this policy is referred to as the *Shortest Expected Remaining Processing Time first (SERPT)* policy.

Theorem 14.34. *The preemptive SERPT policy minimizes the total expected completion time in a two machine open shop in the class of preemptive dynamic policies.*

Proof. Let A_{ij}, $i = 1,2$, $j = 1,\ldots,n$, denote the time that j jobs have completed their processing requirements on machine i. An idle period on machine 2 occurs if and only if

$$A_{1,n-1} \leq A_{2,n-1} \leq A_{1,n}$$

and an idle period on machine 1 occurs if and only if

$$A_{2,n-1} \leq A_{1,n-1} \leq A_{2,n}.$$

Let $j_1, j_2, \ldots, j_n$ denote the sequence in which the jobs leave the system, i.e., job j_1 is the first one to complete both operations, job j_2 the second, and so on. Under the *SERPT* policy

$$C_{j_k} = \max(A_{1,k}, A_{2,k}) = \max\Big(\sum_{l=1}^{k} X_{1l}, \sum_{l=1}^{k} X_{2l}\Big), \qquad k = 1,\ldots,n-1$$

This implies that the time epoch of the kth job completion, $k = 1,\ldots,n-1$, is a random variable which is the maximum of two independent random variables, both with *Erlang(k)* distributions. The distribution of the last job completion, the makespan, is different. It is clear that under the preemptive *SERPT* policy the sum of the expected completion times of the first $n-1$ jobs that leave the system are minimized. It is not immediately obvious that *SERPT* minimizes the sum of all n completion times. Let

$$B = \max(A_{1,n-1}, A_{2,n-1}).$$

The random variable B is independent of the policy. At time B, each machine has at most one more job to complete. A distinction can now be made between two cases.

First, consider the case where, at B, a job remains to be completed on only one of the two machines. In this case, neither the probability of this event occurring nor the waiting cost incurred by the last job which leaves the system (at $\max(A_{1,n}, A_{2,n})$) depends on the policy. Since *SERPT* minimizes the expected sum of completion times of the first $n-1$ jobs to leave the system, it follows that *SERPT* minimizes the expected sum of the completion times of all n jobs.

Second, consider the case where, at time B, a job remains to be processed on both machines. Either (i) there is one job left which needs processing on both machines or (ii) there are two jobs left, each needing processing on one machine (a different machine for each). Under (i) the expected sum of the completion times of the last two jobs to complete their processing is $E(B)+E(B+2)$, while under (ii) it is $E(B)+1+E(B)+1$. In both subcases the expected sum of the completion times of the last two jobs is the same. As *SERPT* minimizes the expected sum of the completion times of the first $n-2$ jobs to leave the system, it follows that *SERPT* minimizes the expected sum of the completion times of all n jobs. □

Unfortunately, no results have been reported in the literature with respect to stochastic open shops with more than 2 machines.

Consider now the two-machine job shop with job j having a processing time on machine 1 which is exponentially distributed with rate λ_j and a processing time on machine 2 which is exponentially distributed with rate μ_j. Some of the jobs have to be processed first on machine 1 and then on machine 2, while the remaining jobs have to be processed first on machine 2 and then on machine 1. Let $J_{1,2}$ denote the first set of jobs and let $J_{2,1}$ denote the second set of jobs. Minimizing the expected makespan turns out to be an easy extension of the two machine flow shop model with exponential processing times.

Theorem 14.35. *The following policy minimizes the expected makespan in the class of nonpreemptive dynamic policies as well as in the class of preemptive dynamic policies: when machine 1 is freed the decision-maker selects from $J_{1,2}$ the job with the highest $\lambda_j - \mu_j$; if all jobs from $J_{1,2}$ have received processing on machine 1 he may take any job from $J_{2,1}$. When machine 2 is freed the decision-maker selects from $J_{2,1}$ the job with the highest $\mu_j - \lambda_j$; if all jobs from $J_{2,1}$ have received processing on machine 2 he may take any job from $J_{1,2}$.*

The result described in *Theorem 14.35* is somewhat similar to the result obtained by Jackson for $J2 \mid\mid C_{\max}$. In deterministic scheduling the research on the more general $Jm \mid\mid C_{\max}$ has focused on heuristics and enumerative procedures. In stochastic scheduling less research has been done on job shops with more than two machines.

14.6. Discussion

No framework or classification scheme has ever been introduced for stochastic scheduling problems. It is more difficult to develop such a scheme for stochastic scheduling problems than for deterministic scheduling problems. For example, it has to be specified which class of policies is considered, it has to be specified whether the processing times of the n jobs are independent or correlated (e.g., equal to the same random variable), it may have to be specified that the processing times are of one type of distribution (e.g., exponential), while the due dates are of another (e.g., deterministic). For these reasons no framework has been introduced in this chapter either.

Table 15.1 outlines a number of scheduling problems of which stochastic versions are tractable. This list refers to most of the problems discussed in this chapter. In the distribution column the distribution of the processing times is specified. If the entry in this column specifies a form of stochastic dominance, then the n processing times are arbitrarily distributed and ordered accordingly. The due dates in this table are considered fixed (deterministic).

Comparing Table 15.1 with the results described in earlier chapters of this book reveals that there are a number of stochastic scheduling problems that are tractable while their deterministic counterparts are NP-hard. The four NP-hard deterministic problems are:

$$\begin{array}{ll} \text{(i)} & 1 \mid r_j, pmtn \mid \sum w_j C_j, \\ \text{(ii)} & 1 \mid d_j = d \mid \sum w_j U_j, \\ \text{(iii)} & 1 \mid d_j = d \mid \sum w_j T_j, \\ \text{(iv)} & P \mid\mid C_{\max}. \end{array}$$

The first problem allows for a nice solution when the processing times are exponential and the release dates are arbitrarily distributed. The optimal policy is then the preemptive WSEPT rule. When the processing time distributions are anything but exponential it appears that the preemptive WSEPT rule is not necessarily optimal. The stochastic counterparts of the second and third problem also lead to the WSEPT rule when the processing time distributions are exponential and the jobs have a common due date which is arbitrarily distributed. Also here, if the processing times are anything but exponential the optimal rule is not necessarily WSEPT.

The stochastic counterparts of $P \mid\mid C_{\max}$ are slightly different. When the processing times are exponential the LEPT rule minimizes the expected makespan in all classes of policies. However, this holds for other distributions also. If the processing times are DCR (e.g., hyperexponentially distributed) and satisfy a fairly strong form of stochastic dominance, the LEPT rule is optimal as well. Note that if preemptions are allowed, and the processing times are DCR, the nonpreemptive LEPT rule remains optimal. Note also, that if the n processing times have the same mean and are hyperexponentially distributed as in Example 14.21, then the LV rule minimizes the expected makespan.

Of course, there are also problems of which the deterministic version is easy and the version with exponential processing times is hard. Examples are:

$$\begin{array}{ll} \text{(i)} & O2 \mid\mid C_{\max}, \\ \text{(ii)} & P \mid p_j = 1, tree \mid C_{\max}. \end{array}$$

For the $O2 \mid\mid C_{\max}$ problem the LAPT rule is optimal; when the processing times are exponential the problem appears to be very hard. For the deterministic problem $P \mid p_j = 1, tree \mid C_{\max}$ the CP rule is optimal. For the version of the same problem with all processing times i.i.d. exponential the optimal policy is not known and may depend on the form of the tree. One would expect that there are many scheduling problems of which the deterministic version with unit processing times is easy, and of which the stochastic version with all processing times i.i.d. exponential is hard.

Table 15.1: Tractable Stochastic Scheduling Problems

DETERMINISTIC COUNTERPART	DISTRIBUTIONS	OPTIMAL POLICY
$1 \mid\mid \sum w_j C_j$	arbitrary	WSEPT
$1 \mid r_j, pmtn \mid \sum w_j C_j$	exponential	WSEPT (preemptive)
$P \mid\mid C_{\max}$	exponential	LEPT
$P \mid pmtn \mid C_{\max}$	exponential	LEPT
$P \mid\mid \sum C_j$	$\geq_{st}$	SEPT
$P2 \mid p_j = 1, intree \mid C_{\max}$	exponential	CP
$P2 \mid p_j = 1, intree \mid \sum C_j$	exponential	CP
$F2 \mid\mid C_{\max}$	exponential	$(\lambda_j - \mu_j)\downarrow$
$F \mid p_{ij} = p_j \mid C_{\max}$	$\geq_{as}$	SEPT-LEPT
$F \mid p_{ij} = p_j \mid \sum C_j$	$\geq_{as}$	SEPT
$O2 \mid p_{ij} = p_j \mid C_{\max}$	exponential	Theorem 14.32
$O2 \mid p_{ij} = 1, pmtn \mid \sum C_j$	exponential	SERPT
$J2 \mid\mid C_{\max}$	exponential	Theorem 14.35