Alcom-FT Technical Report Series
ALCOMFT-TR-03-190

Multicriteria scheduling

Han Hoogeveen *

Department of Computer Science
Utrecht University
P.O. Box 80089, 3508 TB Utrecht, The Netherlands
Email: slam@cs.uu.nl

December 5, 2003

Abstract

In this paper we present a survey of the most important results obtained in multicriteria scheduling.

1 Introduction

Scheduling theory has been developed to solve problems occurring in for instance production facilities. The basic scheduling problem can be described as finding for each of the tasks, which are also called jobs, an execution interval on one of the machines that are able to execute it, such that all side-constraints are met; obviously, this should be done in such a way that the resulting solution, which is called a schedule, is best possible, that is, it minimizes the given objective function. Since the first scheduling paper appeared in 1954, many variants of the basic scheduling problem have been formulated by differentiating between machine environments, side-constraints, and objective functions. Until the late eighties, however, it was common practice that in the objective function only one performance criterion was taken into account. In practice, however, quality is a multidimensional notion. A firm, for instance, judges a production scheme on the basis of a number of criteria, for example, work-in-process inventories and observance of due dates. If only one criterion is taken into account, then the outcome is likely to be unbalanced, no matter what criterion is considered. If everything is set on keeping work-in-process inventories low, then some products are likely to be completed far beyond their due dates, whereas, if the main goal is to keep the customers satisfied by observing due dates, then the work-in-process inventories are likely to be large. In order to reach an acceptable compromise, one has to measure the quality of a solution on all important criteria. This notice has led to the development of the area of multicriteria scheduling.

In this paper, we will review the important results that have been obtained in the field of multicriteria scheduling. In Section 2, we introduce the basic concepts of scheduling in general and bicriteria scheduling in particular. In Section 3, we present two example problems and show

*Supported by EC Contract IST-1999-14186 (Project alcom-FT)

how to solve these. In Section 4 we look at earliness-tardiness scheduling, which is regarded as the most natural bicriteria scheduling problem. In Section 5 we survey results that have been achieved in the area of scheduling with controllable processing times. In Sections 6 and 7 we discuss the remaining results on single-machine and multi-machine scheduling that have appeared in the literature; in the latter section we consider both the parallel machine environment and the shop environment. In Section 8 we discuss the area of simultaneous approximation, and finally in Section 9 we discuss a number of new models. We refer to the books by Pinedo (2002) and T'kindt and Billaut (2002) for an introduction to scheduling theory and for an elaborate discussion of multicriteria scheduling, respectively.

2 Basic concepts and notation

In this section we introduce the basic concepts and the notation that we use. The standard scheduling notation is introduced in Subsection 2.1. In Subsection 2.2 we deal with the basics of bicriteria scheduling and present an extension of the three-field notation scheme of Graham, Lawler, Lenstra, and Rinnooy Kan (1979) to denote multicriteria scheduling problems.

2.1 Basic scheduling concepts

We start with introducing some important notation, where we concentrate on the performance criteria without elaborating on the machine environments etc. We assume that we are given n jobs, which we denote by $J_1, \ldots, J_n$. These jobs are to be scheduled on a set of machines that are continuously available from time zero onwards and that can handle only one job at a time. Depending on the machine configuration, we distinguish between a single-machine scheduling problem, parallel machine scheduling problems and shop models. Job J_j ($j = 1, \ldots, n$) has a *processing time* p_j, which means that it has to be processed for a period of length p_j. Given a schedule σ, we denote the starting time of job J_j in σ by $S_j(\sigma)$, and we use $C_j(\sigma)$ to denote its completion time in σ; we omit the argument σ when it is clear to which schedule we are referring. If preemption is not allowed, that is, we must finish the job without interruption once it has been started, then we have $S_j + p_j = C_j$. The possibilities for executing job J_j may be curtailed by the existence of a *release date* r_j or a *deadline* $\bar{d}_j$, which decree a lower bound on the starting time of J_j and an upper bound on the completion time of job J_j, respectively. Furthermore, for each job J_j a *due date* d_j may be defined, which indicates a preferred completion time. Finally, job J_j may have a weight w_j to express its importance. For a given schedule σ, $L_j(\sigma) = C_j(\sigma) - d_j$ is defined as the *lateness* of job J_j, whereas the *tardiness* $T_j(\sigma)$ of job J_j in σ is defined as $T_j(\sigma) = \max\{L_j(\sigma), 0\}$. Maximum lateness can be generalized to *maximum cost* $f_{\max}$, which is defined as follows. Each job J_j ($j = 1, \ldots, n$) gets its own cost function $f_j(C_j)$, which may be any function, as long as it is nondecreasing in C_j. For a given schedule σ, the maximum cost of σ is defined as $f_{\max}(\sigma) = \max_j f_j(C_j(\sigma))$. We use the indicator function $U_j(\sigma)$ to denote whether job J_j is tardy (then $U_j = 1$) or on time (then $U_j = 0$) in σ. The opposite of tardiness for job J_j is *earliness*, which is defined as $E_j(\sigma) = \max\{d_j - C_j(\sigma), 0\}$. The following performance criteria appear frequently in the literature

maximum completion time or makespan	$C_{\max}(\sigma) = \max_j C_j(\sigma)$
total (weighted) completion time	$\sum_{j=1}^n w_j C_j(\sigma)$
maximum lateness	$L_{\max}(\sigma) = \max_j L_j(\sigma)$
maximum tardiness	$T_{\max}(\sigma) = \max_j T_j(\sigma)$
maximum cost	$f_{\max}(\sigma) = \max_j f_j(C_j(\sigma))$
total (weighted) tardiness	$\sum_{j=1}^n w_j T_j(\sigma)$
maximum earliness	$E_{\max}(\sigma) = \max_j E_j(\sigma)$
total (weighted) earliness	$\sum_{j=1}^n w_j E_j(\sigma)$
(weighted) number of tardy jobs	$\sum_{j=1}^n w_j U_j(\sigma)$

All these criteria except for $E_{\max}$ and $\sum_{j=1}^n w_j E_j$ are *regular*, that is, the value of the objective function cannot be decreased by inserting idle time into the schedule. Although any combination of two criteria out of this list is possible of course, some are more obvious than others. A natural one is to consider a combination of a tardiness and an earliness performance criterion, which reflects the just-in-time objective, as both an early delivery and a tardy delivery are penalized. Before discussing the results on multicriteria scheduling that have appeared in the literature, we will take a closer look at the basic concepts of multicriteria scheduling, which will be illustrated by an example in Section 3, in which we combine $L_{\max}$ and $\sum_j C_j$; this is the combination studied by Smith (1956).

2.2 Basic concepts of multicriteria scheduling

Suppose that we have selected the two performance criteria, say f and g, that we want to take into account. Without loss of generality, we assume that these criteria are to be minimized. Unless we are extremely lucky, there will be no schedule that achieves the minimum value for both performance criteria simultaneously, which implies that we have to give in on the quality of at least one of the two criteria. If one performance criterion, say f, is far more important than the other one, then an obvious approach is to find the optimum value with respect to criterion f, which we denote by f^*, and choose from among the set of optimum schedules for f the one that performs best on g. Such an approach is called *hierarchical optimization* or *lexicographical optimization*: in the first stage, you have to minimize the value of the more important criterion f, whereas in the second stage, the second criterion g is minimized subject to the additional constraint that $f \leq f^*$. Following the notation by T'Kindt and Billaut, we will denote this by putting $Lex(f, g)$ in the third field of the $\alpha|\beta|\gamma$ notation scheme by Graham et al. (1979), where the criterion mentioned first in the argument of Lex is the more important one.

If no criterion is dominant, then lexicographical optimization may lead to a schedule that is unbalanced; the score on the second criterion can be greatly improved while losing only a little on the first criterion. In this case, *simultaneous optimization* may be a better choice. Evans (1984) and Fry, Armstrong, and Lewis (1989) distinguish three different approaches in simultaneous optimization: *a priori optimization*, *interactive optimization*, and *a posteriori optimization*. In case of *a priori optimization*, both criteria are aggregated into one *composite objective function* $F(f(\sigma), g(\sigma))$ for some *given* function F, where σ is the schedule under consideration, after which an optimum solution is determined for this one problem as a whole. This function F can be linear, like for example $\alpha f(\sigma) + g(\sigma)$, where α is a given constant that indicates the relative importance of criterion f with respect to criterion g, but it may just as well be a quadratic or even more exotic function. There are two drawbacks involved in this approach. The first one comes from practice: it is usually unknown what this function F looks like. The person in charge of choosing the eventual production schedule may know which schedule is preferable when presented with a few alternatives, but may find it hard to express this preference in a function F. The second drawback is the computational inaccessibility of minimizing the function $F(f, g)$ in a *direct* manner, especially when the function F is nonlinear. There are two ways to circumvent these drawbacks. The first one requires the active involvement of a decision maker during the solution process. Given one or more already obtained, relevant solutions, the decision maker must indicate which one is preferable, and if not satisfied yet, in which direction the search should continue; this scenario is called *interactive optimization*. The second possibility is to solve the problem in a roundabout way. The idea behind it is that we select from the set of solutions a subset that contains an optimum solution for each *reasonable* composite objective function F that we can think of. If the function F is known, then we compute the optimum solution in this set. If F is not known, then we present this set to the decision maker and let him/her choose; this scenario is called *a posteriori optimization*. Note that a posteriori optimization is clearly the most difficult variant of the three approaches; if we can solve this problem, then we can solve the other two as well, if we may assume that the function F specified by the decision maker beforehand in a priori optimization will be 'reasonable'. Therefore, we assume from now on that we consider the a posteriori variant when we write simultaneous optimization without a further specification. What is left is to specify the properties that a composite objective function must possess to be 'reasonable'. The only restriction that we put on the function F is that it is *nondecreasing in both arguments*.

Definition 1 *The function $F(f, g)$ is said to be nondecreasing in both arguments if for any pair of outcome values (x, y) of the functions f and g, we have*

$$F(x, y) \leq F(x + A, y + B),$$

for each pair of nonnegative values A and B.

This is a natural assumption, since it reflects that we want to minimize both f and g. It has the very convenient effect that under this assumption there exists a *Pareto optimal*, or *nondominated*, schedule by which the optimum is attained.

Definition 2 *A feasible schedule σ is Pareto optimal, or nondominated, with respect to the performance criteria f and g if there is no feasible schedule π such that both $f(\pi) \leq f(\sigma)$ and $g(\pi) \leq g(\sigma)$, where at least one of the inequalities is strict.*

Theorem 3 *If the composite objective function F is nondecreasing in both arguments, then there exists a Pareto optimal schedule that minimizes F.* ∎

A special category of the dominated schedules is formed by the *weak Pareto optimal schedules*, which are defined as follows.

Definition 4 *A feasible schedule σ is a weak Pareto optimal schedule with respect to the performance criteria f and g if it is not Pareto optimal and if there does not exist a schedule π such that both $f(\pi) < f(\sigma)$ and $g(\pi) < g(\sigma)$.*

Usually, the number of Pareto optimal points is finite. When preemption is allowed or if it is advantageous to insert idle time, the number of Pareto optimal points may become infinite. In this case, we would like to have an implicit description of the set of Pareto optimal points by specifying the line-segments that contain the Pareto optimal points together with the isolated Pareto optimal points. The corresponding curve is called the *trade-off curve* by Hoogeveen (1992), and it is defined as the curve that contains all Pareto optimal points.

We illustrate the above by the following single-machine example with performance criteria $L_{\max}$ and $\sum_j C_j$. We are given 4 jobs, the data of which are found in Table 1.

	J_1	J_2	J_3	J_4
p_j	3	4	5	6
d_j	20	16	11	5

Table 1: Processing times and due dates

Since both criteria are regular and F has been assumed to be nonincreasing in both arguments, there is no use in leaving the machine idle if there is work to be done. Hence, a schedule is fully specified by the order in which the jobs are processed. We have enumerated all 24 possible schedules and computed the score on both criteria. The corresponding points are shown in Figure 1, where the bullet, the open circle, and the asterisk indicate that the corresponding schedules are Pareto optimal, weak Pareto optimal, and dominated, respectively.

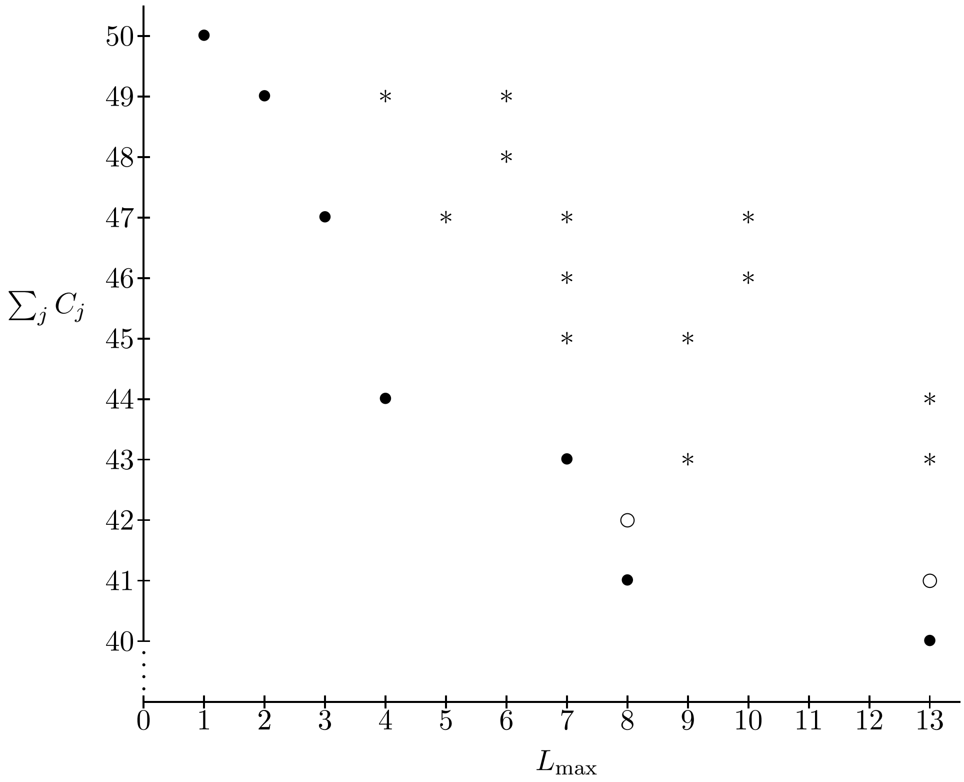

Figure 1: The set of feasible points

Figure 1 shows that if we want to apply lexicographical optimization, then we should select the schedules corresponding to the points $(\sum_j C_j, L_{\max}) = (50, 1)$ and $(40, 13)$, depending on which criterion is the dominant one. If we want to apply a posteriori optimization, then we should present the set of points $(50, 1), (49, 2), (47, 3), (44, 4), (43, 7), (41, 8)$, and $(40, 13)$ to the decision maker, if no further characteristics of F (except for being nondecreasing in both arguments) are known. Sometimes, however, more is known about F. The first possible additional restriction is that for one or both of the criteria an upper bound has been specified, for example that the maximum lateness should be no more than 7. This is easily incorporated by presenting only those Pareto optimal points that satisfy the bounds; this usually leads to a straightforward adjustment of the procedure used to generate the possibly optimal points. The second possible additional restriction is that F is linear. In this case, we can show that the optimum is attained in an *extreme* point by the corresponding *extreme* schedule.

Definition 5 *An extreme point is a vertex of the efficient frontier, which is defined as the lower envelope of the convex hull of the set of Pareto optimal points.*

Note that we need the lower envelope of the convex hull, since we want to minimize $F(f, g)$. Hence, the efficient frontier is a piecewise-linear convex function, where each endpoint corresponds to the solution of one of the lexicographical optimization problems, where each breakpoint is Pareto optimal, and each Pareto optimal point is located either on or above this function.

Theorem 6 *If the composite objective function F is linear, then there exists an extreme schedule that minimizes F.*

Proof. Suppose that the optimum is attained in a point (x, y). If this point is not on the efficient frontier, then there exists a point (x_0, y_0) on the efficient frontier with $x_0 \leq x_1$ and $y_0 \leq y_1$, which implies that $F(x_0, y_0) \leq F(x, y)$. If (x_0, y_0) is not an extreme point, then it must lie on a line-segment that connects two extreme points. Because of the linearity of F, we know that one of these extreme points is at least as good as (x_0, y_0). ■

In our example, the number of Pareto optimal points is finite, and therefore the trade-off curve is not a real curve but just consists of this set of points. The efficient frontier is derived by connecting the points $(50, 1), (44, 4)$, $(41, 8)$, and $(40, 13)$ in Figure 1; these four points are the extreme points. Note that the trade-off curve and the efficient frontier are equal only if the trade-off curve is convex.

Following the notation by T'Kindt and Billaut (2002), we will denote the problem of minimizing a linear composite objective function by putting $F_l(f, g)$ in the third field of the $\alpha|\beta|\gamma$ notation scheme by Graham et al. (1979).

In the above example, the only advantage of our foreknowledge that the objective function is linear comes from the possibility of reducing the number of solutions that we have to present to the decision maker. There are two situations in which it is much more rewarding. The first one occurs if the number of Pareto optimal points is infinite, for instance because the trade-off curve contains a segment in which each point corresponds to a Pareto optimal schedule, which may take place if it is possible to apply preemption or to insert idle time. The second possible advantage occurs when it is computationally hard to find Pareto optimal points but easy to find extreme schedules. Note that the set of extreme points is a subset of the set of *supported points*, which is defined as the set containing all Pareto optimal points that are located on the efficient frontier (T'Kindt and Billaut, 2002).

Now we return to the problem of finding the set of Pareto optimal points. One characteristic of a Pareto optimal point (x, y) is that we can find the value y, given the value x, by minimizing the criterion g subject to the constraint that the value of f amounts to no more than x. If we would know all these values of x that correspond to a Pareto optimal point, then we would simply compute for each of these the corresponding y value. In case of hierarchical optimization, this is a viable approach, since we can compute x by finding the optimum with respect to criterion f. For the remaining x values, there is more work to be done. One approach to achieve this is the following, which is known as the *ϵ-constraint approach* in the terminology of T'kindt and Billaut (2002). In the first step, we apply hierarchical optimization with g as the more important criterion; this yields the first Pareto optimal point, which we denote by (x^1, y^1). Although we do not know the x value of the second Pareto optimal point, which we denote by (x^2, y^2), we do know that $x^2 < x^1$, and if the outcomes are integral then $x^2 \leq x^1 - 1$. We therefore continue our quest by minimizing g subject to the constraint that $f < x^1$ (or $f \leq x^1 - 1$ in case of integral outcomes). The solution of this problem gives us the value y^2, as we will show below. Knowing y^2, we can then find the right value of x^2 by minimizing f subject to the constraint $g \leq y^2$. Knowing (x^2, y^2), we can repeat this approach until we encounter the hierarchical minimum with f as the dominant criterion.

Theorem 7 *Let y be the finite outcome value of the problem of minimizing g subject to the constraint $f \leq \hat{x}$, and let x be the outcome value of the problem of minimizing f subject to the constraint $g \leq y$. Then (x, y) is a Pareto optimal point.*

Proof. Suppose to the contrary that (x, y) is not Pareto optimal; hence, there exists a Pareto optimal point $(\bar{x}, \bar{y})$ that dominates (x, y), which implies that $\bar{x} \leq x$ and $\bar{y} \leq y$. Let π denote the schedule that is found when solving the problem of minimizing f subject to $g \leq y$; let $\bar{\pi}$ be the schedule corresponding to $(\bar{x}, \bar{y})$. Since $g(\bar{\pi}) = \bar{y} \leq y$, we find that $\bar{\pi}$ is a feasible solution to the problem of minimizing f subject to $g \leq y$. Hence, we must have that $x = f(\pi) \leq f(\bar{\pi}) = \bar{x}$, which implies that $x = \bar{x}$. In a similar fashion, we have that $\bar{\pi}$ is a feasible solution for the problem of minimizing g subject to the constraint $f \leq \hat{x}$, since $\hat{x} \geq x = \bar{x}$. Hence, $y = g(\pi) \leq g(\bar{\pi}) = \bar{y}$, which implies that $y = \bar{y}$. ■

To apply this generic approach, we must be able to solve the problems of minimizing f subject to the constraint $g \leq y$ and of minimizing g subject to the constraint $f \leq x$. If only one of these problems is efficiently solvable, for example the first one, then we can apply the approach above with one adjustment. Given a previously obtained point (x^i, y^i), we solve the problem of minimizing f subject to the constraint $g < y^i$ (or $g \leq y^i - 1$ in case of integral solution values), which gives us a schedule π^{i+1}, and we find the next point (x^{i+1}, y^{i+1}) by putting $x^{i+1} = f(\pi^{i+1})$ and $y^{i+1} = g(\pi^{i+1})$. The proof of the above theorem can be used to show that x^{i+1} corresponds to a Pareto optimal point, but y^{i+1} does not have to be minimum given the constraint $f \leq x^{i+1}$; in this way, we can find a series of points (x, y) with equal x value, which therefore are only weak Pareto optimal points. Eventually, we will end up with the whole set of Pareto optimal points by removing the weak Pareto optimal points, but the number of iterations required may go up significantly. Following the notation introduced by T'Kindt and Billaut (2002), we denote this approach by putting $\epsilon(g|f)$ in the third field of the three-field-notation scheme; this should be read as enumerating the Pareto optimal points by solving a series of subproblems of minimizing g subject to a given upper bound on f. Obviously, this approach is viable only if the number of Pareto optimal points is finite.

In Section 3 we will show our generic approach for the problem of minimizing $L_{\max}$ and $\sum_j C_j$ on a single machine. We first summarize the extension of the three-field notation scheme to denote bicriteria scheduling problems.

- $\alpha|\beta|Lex(f,g)$ denotes the problem of minimizing g subject to the constraint that the solution value of f is minimum.
- $\alpha|\beta|F_l(f,g)$ denotes the problem of minimizing a linear composite objective function in f and g with unknown relative weight.
- $\alpha|\beta|F(f,g)$ denotes the problem of minimizing a composite objective function in f and g, where the only foreknowledge is that this function is nondecreasing in both arguments.
- $\alpha|\beta|\epsilon(f|g)$ denotes the problem of finding Pareto optimal points by solving a series of problems of the form minimize f given an upper bound on g. When we are working with the a posteriori variant of simultaneous optimization, we assume that the whole set of Pareto optimal points is enumerated.

3 Two example problems

In this section we consider two bicriteria problems. The first one is $1||F(L_{\max}, \sum_j C_j)$, which serves as an example problem from theory, in which the set of Pareto optimal points is discrete. The second problem is the problem of producing glass bottles in an efficient manner; this is a typical bicriteria problem from industry, in which the set of Pareto optimal points is convex. The glass bottle problem is due to T'kindt, Billaut, and Proust (2001). We start with the simultaneous minimization of $L_{\max}$ and $\sum_j C_j$ on a single machine,

3.1 The $1||F(L_{\max}, \sum_j C_j)$ problem

Before analyzing the problem of minimizing $L_{\max}$ and $\sum_j C_j$ simultaneously in an a posteriori approach, we analyze the two hierarchical minimization problems. We start with the case that $\sum_j C_j$ is the dominant criterion. Smith (1956) showed that $1||\sum_j C_j$ is minimized by arranging the jobs in SPT-order, that is, in order of nondecreasing processing time. The only flexibility that is left for minimizing $L_{\max}$ given that it must be an SPT-schedule is that we can put jobs with equal length in EDD-order, that is, in order of nondecreasing due date (Jackson, 1955). It is easily verified that this solves $1||Lex(\sum_j C_j, L_{\max})$. We denote the corresponding $L_{\max}$ value by $L_{\max}(SPT)$.

If $L_{\max}$ is dominant, then we start by solving $1||L_{\max}$, which is done by putting the jobs in EDD-order; let L^* denote the outcome value. After that, we have to minimize $\sum_j C_j$ subject to the constraint that $L_{\max} \le L^*$, which we denote by $1|L_{\max} \le L^*|\sum_j C_j$. The constraint $L_{\max} \le L^*$ yields a deadline $\bar{d}_j = d_j + L^*$ on the completion time for each job J_j, since

$$L_{\max} \le L^* \iff C_j - d_j \le L^* \quad \forall j \iff C_j \le d_j + L^* \equiv \bar{d}_j \quad \forall j.$$

The problem $1|\bar{d}_j|\sum_j C_j$ is solved in $O(n \log n)$ time by Smith's backward scheduling rule (Smith, 1956): *while there are unscheduled jobs, schedule from among all jobs that can be scheduled last a job with the largest processing time.* This rule is easily validated by an interchange argument.

Note that if there are no deadlines, then this rule comes down to sequencing the jobs in SPT order. This solves our second hierarchical minimization problem.

Since it is easy to solve a problem of the form $1|L_{\max} \leq L|\sum_j C_j$ for any upper bound L on $L_{\max}$ with $L \geq L^*$, we can apply our generic approach. We start with L equal to the maximum $L_{\max}$ value of interest, that is, $L_{\max}(SPT)$. We solve the corresponding $1|L_{\max} \leq L|\sum_j C_j$ problem, which gives us the first Pareto optimal point. Given a (weak) Pareto optimal point with corresponding schedule σ, we then replace L by $L_{\max}(\sigma) - 1$, and proceed in the same way until we find a schedule with $L_{\max} = L^*$. Note that in each step we exclude the previously obtained schedule; note further that, due to the integrality of the data, there is no schedule with $L_{\max}$ value in between $L_{\max}(\sigma) - 1$ and $L_{\max}(\sigma)$, as inserting idle time makes no sense. If we apply this to our example of Section 2, then we see that we start with determining the point $(40, 13)$. We then set $L \leftarrow 12$; solving $1|L_{\max} \leq 12|\sum_j C_j$ brings us in the point $(41, 8)$. We continue by setting $L \leftarrow 7$ and find the point $(43, 7)$, etc.

This approach can be implemented to run in $O(|Q| n \log n)$ time, where $|Q|$ denotes the number of iterations. To bound $|Q|$, we have to address the following two questions:

1. Are all of the points that we find Pareto optimal, or are there weak Pareto optimal points found, too?

2. How many iterations do we need at most?

Although we do not know an algorithm to solve the problem $1|\sum_j C_j \leq C|L_{\max}$ for *any* value of C without enumerating the Pareto optimal points, there is an easy algorithm for the *specific* value of C that corresponds to a Pareto optimal point. All we need to do is to add a tie-breaker to Smith's rule, which then becomes *while there are unscheduled jobs, schedule from among all jobs that can be scheduled last a job with the largest processing time, where in case of a tie pick the job with largest due date.* From now on, we assume this adjustment to be incorporated in Smith's rule.

Theorem 8 *Let L be any value greater than or equal to L^*. The schedule σ determined by solving $1|L_{\max} \leq L|\sum_j C_j$ through Smith's rule yields a Pareto optimal point.* ■

Since we find a Pareto optimal point in each iteration, we only have to find a bound on the number of Pareto optimal points to know the running time of the algorithm. The problem of finding a good upper bound on the number of Pareto optimal points has long been open. Van Wassenhove and Gelders (1980) applied the approach above to find all Pareto optimal points, and they conjectured that the number of Pareto optimal points could be pseudo-polynomial. Hoogeveen and Van de Velde (1995) show that the number of Pareto optimal points is at most equal to $n(n-1)/2 + 1$, which yields a running time of $O(n^3 \log n)$. We will briefly discuss their proof technique.

The proof is based on finding a function such that each Pareto optimal point gets a different number according to the characteristics of the corresponding schedule. To that end, we define the indicator function $\delta_{ij}(\sigma)$ as

$$\delta_{ij}(\sigma) = \begin{cases} 1, & \text{if } C_i(\sigma) < C_j(\sigma) \text{ and } p_i > p_j, \\ 0, & \text{otherwise ,} \end{cases}$$

and $\Delta(\sigma) = \sum_{i,j} \delta_{ij}(\sigma)$. Note that $\delta_{ij}(\sigma) = 1$ implies that the interchange of the jobs J_i and J_j in σ will decrease the total completion time. In that respect, $\delta_{ij}(\sigma) = 1$ signals a *positive* interchange. Observe that $\Delta(SPT) = 0$ and $\Delta(\sigma) \leq n(n-1)/2$ for any feasible schedule σ. In addition, we define a *neutral* interchange with respect to σ as the interchange of two jobs J_i and J_j with $p_i = p_j$.

Lemma 9 *Let σ and π denote two schedules that are generated through Smith's rule. If schedule π can be obtained from schedule σ through a positive interchange, then $\Delta(\pi) < \Delta(\sigma)$.* ■

Theorem 10 *Consider two arbitrary Pareto optimal schedules π and σ. If $\sum_j C_j(\pi) < \sum_j C_j(\sigma)$, then schedule π can be obtained from schedule σ by using positive and neutral changes only, which implies that $\Delta(\pi) < \Delta(\sigma)$.* ■

Theorem 11 *The number of Pareto optimal schedules is bounded by $n(n-1)/2 + 1$, and this bound is tight.*

Proof. The first part follows immediately from Theorem 10. For the second part, consider the following instance of $1||F(\sum_j C_j, L_{\max})$: there are n jobs with processing times $p_j = n - 2 + j$ and due dates $d_j = \sum_{i=j}^{n} p_i + n - j$, for $j = 1, \ldots, n$. A straightforward computation shows that this set of data leads to $n(n-1)/2 + 1$ Pareto optimal schedules. ■

3.2 Producing glass bottles

The second example problem is due to T'kindt, Billaut, and Proust (2001). The problem concerns the optimal plan for the production of glass bottles; it is briefly described as follows (for a full description, we refer to T'kindt et al.). The factory produces n different types of bottles. In a given period, it has to produce b_j bottles of type j ($j = 1, \ldots, n$). There are m machines available; in one time unit, machine M_i can produce k_{ij} bottles of type j, where $k_{ij} > 0$ for all i, j. Producing one bottle of type j on machine M_i yields a profit of c_{ij}. Obviously, maximizing total profit is one criterion. The second criterion comes from balancing the work load. Given a production plan, we denote the time that machine M_i has finished the work that has been assigned to it by $\bar{C}_i$ (we use the 'bar' to distinguish from the job completion time); the time at which all machines are ready is equal to the makespan $C_{\max}$, which should not exceed a given length T. The idle time I_i of machine M_i is then equal to $C_{\max} - \bar{C}_i$; the maximum idle time $I_{\max}$ is used as a measure of the *balancedness* of a production schedule. The second criterion is to minimize the maximum idle time. Another important problem-characteristic is that it is not necessary to produce all bottles of the same type on one machine: changing from the production of one type of bottle to another takes a negligible amount of time. Due to the usage of expensive equipment, however, the same type of bottle can be produced by at most one machine at the same time.

In scheduling terms, the problem described above boils down to a problem on m *unrelated parallel* machines, where preemption is allowed. In this case the problem of minimizing the makespan is solvable in polynomial time through an algorithm by Lawler and Labetoulle (1978). This algorithm consists of two stages: in the first stage, a linear program is solved to decide for each job J_j ($j = 1, \ldots, n$) the fraction of it that must be executed by machine M_i ($i = 1, \ldots, m$), after which in the second stage this assignment of work to the machines is translated into a feasible schedule such that no job is executed by two machines at the same time.

T'kindt et al. apply a similar solution method to solve the glass bottle problem. In the first step, they formulate the problem as a linear programming problem in the following way. Here the decision variables x_{ij} represent the number of bottles of type j that are produced by machine M_i; $\bar{C}_i$ and I_i denote the completion time and the idle time of machine M_i; and $C_{\max}$ and $I_{\max}$ represent the makespan and the maximum idle time. The objective function is chosen as a linear combination of the total gain and the maximum idle time with relative weight α.

$$\min \quad -\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \alpha I_{\max}$$

subject to the constraints

$$\sum_{i=1}^{m} x_{ij} = b_j, \quad \forall j = 1, \dots, n, \tag{1}$$

$$\sum_{i=1}^{m} \frac{x_{ij}}{k_{ij}} \le C_{\max}, \quad \forall j = 1, \dots, n, \tag{2}$$

$$\sum_{j=1}^{n} \frac{x_{ij}}{k_{ij}} = \bar{C}_i, \quad \forall i = 1, \dots, m, \tag{3}$$

$$C_{\max} = I_i + \bar{C}_i, \quad \forall i = 1, \dots, m, \tag{4}$$

$$I_{\max} \ge I_i, \quad \forall i = 1, \dots, m,$$

$$C_{\max} \le T,$$

$$x_{ij} \ge 0, \quad \forall i = 1, \dots, m, \forall j = 1, \dots, n,$$

$$I_i \ge 0, \quad \forall i = 1, \dots, m.$$

Constraint 1 states that each job should be fully executed, and constraint 2 forces that the makespan is big enough to process each job without overlap; these are two of the three constraints in the linear program by Lawler and Labetoulle. The third of their constraints is that the makespan is no less than the maximum machine load, which is enforced by the constraints 3, 4, and the nonnegativity of the I_i variables. Hence, the solution of the above linear program can be turned into a feasible schedule. Note that we have removed the integrality constraints on the x_{ij} variables, which is justified because of the large number of bottles that have to be produced.

Now we come to the objective function. If we want to solve the lexicographical problem, then we simply make the weight of the more important criterion much bigger than the weight of the other criterion, and we are done; in this case, we let α either go to zero or to infinity. The problem with a linear composite objective function with known α and the problem of minimizing one criterion subject to a given bound on the other criterion is easy to incorporate, too. In all other cases, we need the trade-off curve or at least the efficient frontier. By increasing the ratio of the relative weight α, which is known as *parametric analysis* in the theory of linear programming (see for example Bazaraa, Jarvis, and Sherali (1990)), we find a set of production plans that are optimal on some interval for α together with the intervals these plans are optimal on. Now each of these production plans corresponds to an extreme point; connecting the consecutive extreme points yields the efficient frontier, where the gradient of the line segment between two consecutive extreme points is equal to the common α value in their intervals. Moreover, the efficient frontier is identical to the trade-off curve in this case, since each point on the efficient frontier corresponds to a feasible production plan, as can be shown in the following way. Let $(\bar{C}, \bar{I})$ be any point on the efficient frontier; suppose that it is located on the line segment that connects the two extreme points (C^k, I^k) and (C^{k+1}, I^{k+1}). Furthermore, let α^k be the gradient of the line segment, and let y^k and y^{k+1} be the corresponding solutions of the linear programming problem. Since the feasible region of the linear programming problem is convex, the point y on the segment between y^k and y^{k+1} with $I_{\max}$ value equal to $\bar{I}$ corresponds to a feasible solution of the linear programming problem. Let C denote the score of y on the 'total gain' criterion; we need to show that $C = \bar{C}$. Since the value of the objective function in y^k and y^{k+1} is the same if $\alpha = \alpha^k$, and since y is a convexcombination of y^k and y^{k+1}, the value of the objective function in y is equal to that in y^k. Hence, $(C, \bar{I})$ must be on the line segment between (C^k, I^k) and (C^{k+1}, I^{k+1}), which implies that $C = \bar{C}$.

Since the number of Pareto optimal points is infinite, a result by Schrijver (see Hoogeveen, 1996A) implies that there exist pathological cases in which the problem of minimizing a general composite objective function is $\mathcal{NP}$-hard. In general, however, having a compact representation of the trade-off curve will enable a decision maker to choose a very good solution. Obviously, the input of a decision maker during the search process, the so-called *interactive optimization* may greatly reduce the number of intermediate solutions that are inspected before finding the optimum.

4 Earliness-tardiness scheduling

Probably the most appealing bicriteria scheduling problem is 'the' earliness-tardiness problem. The justification comes from the 'just-in-time' philosophy in management theory: the best time for getting an item delivered is the time at which it is needed, which time is given by the due date. As a result, when scheduling the production process we are unhappy with both an early completion, which results in the need to store the product until it can be shipped, and a late completion of a product. In the following, we assume that the due dates have been determined already, which is opposite to the due date assignment model, in which one has the freedom to determine the optimal due date (at a certain cost); we refer to the survey by Gordon, Proth, and Chu (2002) for an overview of the results that have appeared in this area. We distinguish between problems with general due dates and problems with a common due date.

4.1 General due dates

In this subsection we discuss the results that have been obtained in the area of earliness-tardiness scheduling with general due dates. The first issue is of course how to penalize earliness and tardiness. We start with results for minimizing $E_{\max}$ and $T_{\max}$.

Minimizing maximum earliness and maximum tardiness

The first result involving $E_{\max}$ and $T_{\max}$ is due to Garey, Tarjan, and Wilfong (1988). They study a single-machine scheduling problem in which each job J_j has been given a target starting time $d_j - p_j \geq 0$ and a target completion time d_j; the goal is to find a schedule such that each job is executed as close to its preferred execution interval as possible. They present an $O(n \log n)$ algorithm to check whether for a given threshold value y there exists a feasible schedule such that each job is executed in the interval $[d_j - p_j - y, d_j + y]$. By applying binary search, they find the minimum such value y, which is equal to $\max\{E_{\max}, L_{\max}\}$; this algorithm can be implemented to run in time $O(n(\log n + \log p_{\max}))$.

The problem of minimizing a combination of $E_{\max}$ and $T_{\max}$ on a single machine has also been studied by Hoogeveen (1996A). He generalizes $E_{\max}$ to the criterion of maximum promptness $P_{\max}$. Here $P_j = \max\{s_j - S_j, 0\}$, where s_j is a preferred start time; putting s_j equal to $d_j - p_j$ for all $j = 1, \ldots, n$ transforms $P_{\max}$ into $E_{\max}$. Since the constraint $P_{\max} \leq P$ leads to a release date $r_j = s_j - P$ $(j = 1, \ldots, n)$, the problem $1|P_{\max} \leq P|T_{\max}$ is $\mathcal{NP}$-hard in the strong sense, unless there is a relation between s_j and d_j. Hoogeveen assumes therefore that $s_j \in [d_j - p_j - A, d_j - A]$ for all $j = 1, \ldots, n$, where A is any constant. Under this assumption, the $1|P_{\max} \leq P, nmit|L_{\max}$ problem, where the acronym *nmit* indicates that machine idle time is not allowed, is solvable in $O(n \log n)$ time. Using the ϵ-constraint approach the whole set of at most n Pareto optimal points can be identified in $O(n^2)$ time if machine idle time is not allowed. Moreover, using the algorithm for the 'no machine idle time' problem as a subroutine, the trade-off curve can be identified in $O(n^2 \log n)$ time for the case in which machine idle time is allowed. Hence, a problem like $1||E_{\max} + L_{\max}$ is solvable in $O(n^2 \log n)$ time. If we increase the due dates by a large constant, such that each job is early, then we get $E_{\max} = -L_{\min}$, and hence we can solve in $O(n^2 \log n)$ time the problem of *minimizing the range of lateness*, since this problem is not influenced by a shift of the due dates.

Sourd (2001) considers a generalization of the above problem: maximum tardiness is generalized to maximum cost $f_{\max}$ and maximum promptness is generalized to $h_{\max} = \max h_j(S_j)$, where each of the functions h_j is assumed to be nonincreasing in S_j. Obviously, the nonpreemptive problem is $\mathcal{NP}$-hard in the strong sense, but the preemptive version can be attacked using the ϵ-constraint approach. It takes $O(n^2)$ time to compute a Pareto optimal point, but unfortunately, the number of Pareto optimal points can be unbounded. The author shows that a special case that occurs in job shop scheduling is solvable in $O(n^3 \log n)$ time.

Minimizing a function of total earliness and total tardiness

Much more work has been done on the problem of minimizing some function of total earliness and total tardiness with unequal due dates. As Du and Leung (1990) have shown that $1||\sum_j T_j$ is $\mathcal{NP}$-hard (in the ordinary sense), all the earliness-tardiness problems without simplifying assumptions are $\mathcal{NP}$-hard, too. Moreover, specifying an order in which to execute the jobs is not enough now, since it may be advantageous to insert idle time in the schedule. Some researchers have focused on solving this subproblem. We first discuss these results.

1. Finding the optimal starting times

The first result is due to Garey, Tarjan, and Wilfong (1988), who show that the problem of finding the set of starting times that minimizes $\sum_j (E_j + T_j)$ for a given ordering of the jobs on a single machine is solvable in $O(n \log n)$ time. This algorithm works as follows. Suppose that the jobs have been renumbered according to the given ordering. Given an optimal schedule for the first j jobs, the optimal schedule for the first $j + 1$ jobs is determined using a shifting argument: job J_{j+1} initially is assigned to a starting time equal to the maximum of its preferred starting time $d_{j+1} - p_{j+1}$ and the current completion time of job J_j. If job J_{j+1} is started after its preferred starting time, then the last set of contiguously executed jobs (including jobs J_j and J_{j+1}) is shifted towards zero, until, either one of the jobs in this block gets started at its preferred starting time, or the block cannot move to the left any further without pushing an earlier block to the left or starting before time zero, whichever happens first. In all cases, an optimal subschedule for the first $j + 1$ jobs has been obtained, after which the procedure is repeated until all jobs have been scheduled. Garey et al. show that this algorithm can be generalized to solve the problem of minimizing $\sum_j w_j(E_j + T_j)$ for a given ordering of the jobs. The only difference here is that the stopping criteria mentioned above are not sufficient anymore, and we must check if more can be gained by shifting the last block further back (together with the preceeding block if it hits this one). The algorithm can, however, be implemented such that it still runs in $O(n \log n)$ time.

After this, many researchers have investigated extensions. The first extension is to consider convex cost functions $f_j(C_j)$, which are minimum when job J_j completed at its due date d_j. Since the sum of a finite number of convex functions is convex again, the shifting approach by Garey et al. can be applied with a small modification: the last block is shifted back until, either it collides with the previous block, or the cost of the block as a whole is minimized. In the former case, the blocks are merged and the procedure is repeated. Chrétienne and Sourd (2003) have considered the above problem in the more general setting in which a non-bottleneck machine must process a set of jobs subject to precedence constraints between the jobs such that the total cost is minimized, where the cost of a job is any convex function of its completion time (the single-machine case can be modelled by specifying the precedence constraints to force the desired order of the jobs). Chrétienne and Sourd (2003) also consider the case in which the cost of a job J_j is equal to $\alpha_j E_j + \beta_j T_j$; they show that the resulting problem is solvable in $O(n \max\{n, |A|\})$, where $|A|$ denotes the number of arcs in the precedence graph. They show that the special case in which the precedence graph forms a tree is solvable in $O(n^2)$ time and that the problem in which the precedence graph consists of a set of chains is solvable in $O(n \log n)$ time.

A more general contribution is due to Wennink (1995), who models the problem of finding the optimal set of starting times as a max cost flow problem; in Wennink's approach the number of machines involved can be arbitrary. This approach was originally derived as a subroutine for a job shop problem with a general *nonregular cost function*; it is a natural extension of the *longest path* algorithm that is used to find the optimal starting times of the operations in case of a *regular* cost function like makespan given the execution order of the operations on each machine. Moreover, Wennink's algorithm can handle precedence relations between jobs that are executed on different machines. Therefore, it can also be used in case of a machine with infinite capacity, like in the area of resource constrained project scheduling. We describe Wennink's approach below.

As mentioned before, the maximum cost flow problem forms the basis of Wennink's algorithm. This problem can be solved by an efficient combinatorial algorithm (see, for instance, Ahuja, Magnanti, and Orlin (1993)). The linear programming formulation of a max cost flow problem in a graph $G = (V, A)$ is as follows

$$\begin{array}{lll} \max & \sum\limits_{(i,j)\in A} c_{ij} x_{ij} & \\ \text{s.t.} & \sum\limits_{(g,i)\in A} x_{gi} - \sum\limits_{(i,j)\in A} x_{ij} = w_i, & \forall i \in V, \\ & 0 \le x_{ij} \le h_{ij} \ , & \forall (i,j) \in A. \end{array}$$

Here, x_{ij} denotes the flow through arc $(i,j) \in A$, which has a given capacity h_{ij} and cost c_{ij} per unit. Moreover, w_i is the amount of flow that is desired by vertex $i \in V$. The dual of this linear program is the linear programming formulation that we are going to use to model the problem of finding the optimal starting times

$$\min \ \sum_{i\in V} w_i S_i + \sum_{(i,j)\in A} h_{ij} \alpha_{ij}$$

$$
\begin{array}{lll}
\text{s.t.} & S_j - S_i + \alpha_{ij} \geq c_{ij} \ , & \forall (i,j) \in A, \\
& S_i \text{ unrestricted} \ , & \forall i \in V, \\
& \alpha_{ij} \geq 0 \ , & \forall (i,j) \in A.
\end{array}
$$

The variables in this linear program are S_i, for all $i \in V$, and α_{ij}, for all $(i,j) \in A$. Obviously, we will use the variables S_i to model the starting times. Therefore, we build the graph $G = (V, A)$ such that the vertices in V correspond to the jobs $J_1, \ldots, J_n$, to which we add a dummy job J_0. We use the constraint $S_j - S_i + \alpha_{ij} \geq c_{ij}$ to model a relation between the starting times of jobs J_i and J_j. In this constraint, the variable α_{ij} denotes a violation of the constraint $S_j - S_i \geq c_{ij}$, which is penalized by the factor h_{ij}; if a violation is not allowed, then we simply put $h_{ij} = \infty$. The inclusion of violations in the model makes it possible to model more general convex cost functions. For technical reasons, we need that $\sum_{i \in V} w_i = 0$, which is easily achieved by assigning the correct value to w_0. As a result, we may assume without loss of generality that $S_0 = 0$.

We can use the constraint $S_j - S_i + \alpha_{ij} \geq c_{ij}$ to model for instance the ordinary precedence constraint that job J_i should precede job J_j (then $c_{ij} \leftarrow p_i$) or a minimum delay constraint. We can model a release date r_i of job J_i through the constraint $S_i - S_0 \geq r_i$ (or $S_i - S_0 + \alpha_{0i} \geq r_i$ if we allow it to be violated at a given cost h_{0i}). Note that the graph $G = (V, A)$ does not have to be acyclic. A cycle in the graph implies that the difference between the starting times S_j and S_i is not only bounded from below, but also from above. Hence, we can model for instance a deadline $\bar{d}_i$ of job J_i, by including the constraint $S_0 - S_i \geq p_i - \bar{d}_i$, but it can also be used to model a no-wait constraint.

Now we come to the objective function. Obviously, the term $\sum\limits_{i \in V} w_i S_i$ can be used to model the weighted sum of starting times or the weighted sum of completion times. We can model the problem of minimizing the makespan by adding a dummy job J_t and a constraint $S_t - S_i \geq p_i$ for each job J_i without successors (except for J_t), as minimizing S_t renders the makespan then. In a similar fashion, we can model for instance the maximum lateness by adding a constraint $S_t - S_i \geq p_i - d_i$ for each job J_i. Moreover, we can model total weighted tardiness by adding for each job J_i a specific dummy job J_{it}; including the constraints $S_{it} - S_i \geq p_i - d_i$ and $S_{it} \geq 0$ then sees to it that S_{it} becomes equal to the tardiness of job J_i, etc.

The second term of the objective function, $\sum\limits_{(i,j)} h_{ij} \alpha_{ij}$, offers even more powerful modeling capacities. By an appropriate choice of additional constraints, we can model any cost function f that is convex piecewise linear in the starting time of a job. Moreover, we can also model cost functions that are convex and piecewise linear in $S_j - S_i$. We refer to Wennink (1995) for an elaborate discussion of these modeling issues.

Sourd (2002) considers a problem setting in which there are *time windows* and *breaks*. The existence of a set of time windows implies that the job should be entirely executed in one of these; a break signals that the execution of a job is halted during the break but can be continued afterwards. The goal is to find optimal starting times for a given sequence of jobs, where the objective function consists of general completion costs and idle period penalties. Sourd (2002) shows that this problem is solvable through dynamic programming when the cost functions are piecewise linear; convexity is not necessary. In a follow-up paper Sourd (2003B) studies the problem of finding for a given sequence of jobs the set of starting times that correspond to the set of Pareto optimal points for the criteria of punctuality cost and idle time cost. Here punctuality cost is a performance measure that generalizes the earliness/tardiness cost by including, for instance, nonregular delivery costs; this cost function is assumed to be piecewise linear. Sourd

(2003B) shows that the problem can be solved by applying dynamic programming, in which techniques from computational geometry have been incorporated.

2. Finding the optimal sequence of jobs

Since the problem of finding the optimal starting times for a given sequence is solvable in polynomial time, we 'only' have to find the optimal sequence to find an optimal schedule. If we do not insist on finding an optimal solution, then an obvious approach is to apply decomposition: given a sequence the optimal starting times are determined, after which this sequence is altered by applying local search. This approach has for instance been followed by Wan and Yen (2002) for the slightly more general problem $1||\sum_j(\alpha_j P_j + \beta_j T_j)$, where $[s_j, d_j]$ gives the preferred execution interval.

In the remainder of this subsection we discuss exact approaches. As the problem of minimizing total tardiness is $\mathcal{NP}$-hard, any problem that includes $1||\sum_j T_j$ as a subproblem is $\mathcal{NP}$-hard as well. Some special cases have been shown to be solvable in polynomial time. Garey, Tarjan, and Wilfong (1988) show that the EDD-order solves the $1|p_j = p|\sum_j(E_j + T_j)$ problem; this result remains valid for the problem $1|p_j = p|\sum_j(\alpha E_j + \beta T_j)$ problem. Verma and Dessouky (1998) show that the problem $1|p_j = p|\sum_j(\alpha_j E_j + \beta_j T_j)$ is solvable in polynomial time if the weights are *agreeable*, that is, the jobs can be renumbered such that

$$\alpha_1 \leq \alpha_2 \leq \ldots \leq \alpha_n \quad \text{and} \quad \beta_1 \leq \beta_2 \leq \ldots \leq \beta_n;$$

this covers the case with symmetric weights $\alpha_j = \beta_j$. Their algorithm is based on linear programming. They first enumerate the possible completion times of the jobs and then reformulate the problem as one of assigning jobs to completion times, subject to a set of constraints to enforce that at most one completion time is used from each set of overlapping completion times. They show that, under the assumption of agreeable weights, the optimal solution of the LP-relaxation is integral and hence solves the scheduling problem.

The first attempt to solve the general problem was made by Fry and Keong Leong (1987), who solved the $1||\sum_j(\alpha E_j + \beta T_j)$ problem as an integer linear programming problem; this was viable for instances with up to 12 jobs. Kim and Yano (1994) applied branch-and-bound to solve the $1||\sum_j(E_j + T_j)$ problem based on two lower bounds; they can solve instances with up to 20 jobs. A similar result has been reported by Hoogeveen and Van de Velde (1996), who apply branch-and-bound to solve the $1||\sum_j(\alpha E_j + \beta T_j)$ problem. They propose six lower bounds and a number of dominance rules, but they cannot solve instances with size over 25 jobs. Recently, better results have been reported by Sourd and Kedad-Sidhoum (2003) and Bülbül, Kaminsky, and Yano (2001). The first paper considers the problem $1||\sum_j(\alpha_j E_j + \beta_j T_j)$, whereas Bülbül et al. consider the problem $1|r_j|\sum_j(\alpha_j E_j + \beta_j T_j)$. In both papers the problem is formulated as an integer linear programming problem using a time-indexed formulation. Each job J_j is split into p_j unit duration segments, where one of the constraints signals that the segments forming job J_j $(j = 1, \ldots, n)$ must be assigned to consecutive intervals. Next, these consecutivity constraints are relaxed to obtain an assignment problem, where portions of jobs have to be assigned to execution intervals of length 1; assigning one unit of job J_j to period k (that is, the interval $[k-1, k]$) will cost c_{jk} then. This yields the problem of finding a good value for c_{jk} that yields a tight lower bound. Sourd and Kedad-Sidhoum formulate the constraint that

$$\sum_{k=t+1}^{t+p_j} c_{jk} = f(t + p_j) \quad \forall t = 0, \ldots, T - p_j; j = 1, \ldots, n,$$

where $f(t + p_j)$ denotes the cost of completing job J_j at time $t + p_j$, and T is the time horizon, which can be put equal to the latest reasonable completion time of a job. These constraints do not impose a unique solution, and this freedom could be applied to increase the lower bound. The authors report, however, that this maximization is too time-consuming and that choosing a solution such that $c_{jk} = 0$ for $k = d_j - p_j + 2, \ldots, d_j - p_j$ works well in practice. Bülbül, Kaminsky, and Yano (2001) define

$$c_{jk} = \begin{cases} \frac{\alpha_j}{p_j}[(d_j - p_j/2) - (k - 1/2)] & \text{if } k \leq d_j \\ \frac{\beta_j}{p_j}[(k - 1/2) - (d_j - p_j/2)] & \text{if } k > d_j \end{cases}$$

This choice is motivated by the observation that $d_j - (p_j - 1)/2$ is the mean completion time of the segments of job J_j, when it would be executed in the interval $[d_j - p_j, d_j]$. When the c_{jk} values have been determined, an assignment problem remains, in which $\sum_j p_j$ time units have to be assigned to T positions. Sourd and Kedad-Sidhoum design an algorithm for this problem that runs in $O(n^2T)$ time. Bülbül et al. argue that the solution to the assignment problem can easily be converted into a feasible schedule, which yields a good heuristic. Sourd and Kedad-Sidhoum use the lower bound in a branch-and-bound approach. They further incorporate a set of dominance rules. Their branch-and-bound approach solves all instances with up to 30 jobs, where the processing times are drawn from $U[1, 100]$, which denotes the uniform distribution with limits 1 and 100. The hardest instances, which are the ones where the due dates all lie in the interval $[0.6 \sum_j p_j, 1.2 \sum_j p_j]$, are solved in 400 sec. on average. Bülbül et al. do not present a specific branch-and-bound algorithm, but solve the time-indexed formulation using CPlex. For $n = 40$, all instances are solved to optimality in at most 410 seconds, where the processing times are drawn from $U[1, 50]$. They further report on the quality of the lower bound and the quality of the heuristics, which all seem to do fine.

The lower bounds by Sourd and Kedad-Sidhoum (2003) and Bülbül et al. (2001) are obtained by allowing preemption of job J_j on integral time points only. Sourd (2003A) discusses the consequences of allowing preemption at any time point. This gives more freedom to choose the cost of processing job J_j at time t, which yields a better lower bound. This lower bound is computed by approximating the optimal solution of the dual problem by a subgradient method, which yields a lower bound in each iteration. Computing this 'continuous' lower bound is more time-consuming than the computation of the 'discrete' lower bound by Sourd and Kedad-Sidhoum. When the branch-and-bound algorithm by Sourd and Kedad-Sidhoum is run with the continuous lower bound instead of the discrete one, the running time is comparable, although the number of nodes is a factor 5 to 20 smaller, depending on the number of iterations allowed to the subgradient method. The branch-and-bound algorithm with the continuous lower bound by Sourd (2003A) is less sensitive to an increase in processing times though.

Bülbül, Kaminsky, and Yano (2003) use their approach to find a heuristic solution for an m-machine flow shop problem with earliness, tardiness, and holding cost, where the holding cost is computed on basis of the time a job J_j spends in the buffers between the machines. They formulate this problem as an integer linear programming model using the time-indexed formulation. This formulation is relaxed using Dantzig-Wolfe decomposition and Lagrangean relaxation, where the algorithm by Bülbül, Kaminsky, and Yano (2001) for the single machine problem is used to solve the subproblems. Finally, the solution of the relaxed problem is converted into a feasible solution. Results are reported with up to 4 machines and 30 jobs, where processing times are drawn from $U[1, 10]$.

4.2 Common due date scheduling

In this subsection, we assume that the due date d_j of job J_j is equal to the *common due date* d, for $j = 1, \ldots, n$. In the three-field notation scheme, this is denoted by including $d_j = d$ in the second field. Depending on the size of d in comparison to the total processing time, the common due date is called, either *large*, or *small*. In the former case, we assume that the unavailability of the machine before time zero does not complicate the scheduling problem. In the latter case, the size of d is such that increasing d may lead to a better schedule. Therefore, these two subcases are also called *nonrestrictive* and *restrictive*.

Equal job weights, d large

The area of common due date scheduling was introduced by Kanet (1981). Since then, it has received a lot of attention, mainly due to the rich structure possessed by optimal schedules. Kanet looked at the problem $1|d_j = d|\sum_j(E_j + T_j)$; this objective function can be rewritten as $\sum_j |C_j - d|$, that is, the total deviation from the common due date d. He presents an algorithm to solve the problem with a large common due date; this algorithm can be implemented to run in $O(n \log n)$ time. Hall (1986) and Emmons (1987) discovered for this problem the nice structure that we will describe below. It is easily verified (see for instance the review paper by Baker and Scudder (1990)) that there exists an optimal schedule such that

- The jobs are processed contiguously;
- There is a job that is completed at time d.

As a consequence, the cost of a schedule that satisfies these properties can be expressed as a linear function of the processing times as follows. First look at the jobs that complete at or before time d; suppose this part of the schedule consists of the jobs $J_{[1]}, \ldots, J_{[m]}$, where $J_{[1]}$ is the first job in the schedule and $J_{[m]}$ is the job that completes at time d. An easy computation shows that $|C_{[j]} - d| = \sum_{h=j+1}^{m} p_{[h]}$, for $j = 1, \ldots, m$, and hence

$$\sum_{j=1}^{m} |C_{[j]} - d| = \sum_{j=1}^{m} \sum_{h=j+1}^{m} p_{[h]} = \sum_{j=1}^{m} (j-1)p_{[j]}.$$

A similar result can be derived for the jobs that complete after time d: if the jobs $J_{[m+1]}, \ldots, J_{[n]}$ are executed in the tardy part of the schedule in this order (that is, $J_{[n]}$ is the last job), then

$$\sum_{j=m+1}^{n} |C_{[j]} - d| = \sum_{j=m+1}^{n} \sum_{h=m+1}^{j} p_{[h]} = \sum_{j=m+1}^{n} (n+1-j)p_{[j]}.$$

This observation implies that the problem of minimizing the total deviation from d can be solved as an assignment problem, where jobs are assigned to positions. The cost of assigning job J_j to early position h in the schedule is equal to $(h-1)p_j$; assigning it to the hth tardy position, counted from behind, is equal to hp_j. Therefore, we say that the hth early position has *positional weight* $h - 1$, and the hth tardy position (counted from behind) has *positional weight* h. Since there are at most n early and at most n tardy positions, we proceed by selecting the n positions with smallest positional weight, where ties are broken arbitrarily; the correctness of this approach is easily verified by an interchange argument. What is left to do is to assign the n jobs to the n selected positions. Assume that the jobs have been renumbered such that

$p_1 \geq p_2 \geq \ldots \geq p_n$. We solve the resulting assignment problem by assigning p_j to the position with the jth smallest positional weight, where ties are broken arbitrarily. For Kanet's problem, this boils down to assigning J_1 to the first early position, then jobs $\{J_2, J_3\}$ to the second early position and the first tardy position, and so on, where it is irrelevant which of the two jobs in the pair goes to the early or tardy part of the schedule. We can exploit this freedom of breaking ties to minimize the total length of the early part of the schedule. This yields the observation by Bagchi, Sullivan, and Chang (1986) that the minimum value of d such that it is nonrestrictive is equal to $p_1 + p_3 + \ldots + p_n$ if n is odd and $p_1 + p_3 + \ldots + p_{n-1}$ if n is even. Since the jobs in the early part of the schedule are in LPT-order and the ones in the tardy part of the schedule are in SPT-order, the schedule is said to possess a *V-shape* structure.

The assignment approach can be used to solve several more cases. The first one is that there are parallel machines, which can be either identical or uniform. The second extension is to generalize the objective function to $\sum_j(\alpha E_j + \beta T_j)$. Furthermore, it can be used to solve a further generalization of the objective function with a linear term like $\gamma T_{\max}$ or $\lambda \sum_j C_j$ (Hoogeveen and Van de Velde (1997)). In all cases, we have to compute the positional weights, select the n positions with smallest weight, and assign the jobs in the correct way. Hoogeveen and Van de Velde (1997) further show that a generalization of the common due date problem, which is called the *almost common due date* problem is solvable in $O(n^2)$ time. This is the $1||\sum_j(\alpha E_j + \beta T_j)$ problem, where the due dates and processing times are such that $d_j \in [d, d+p_j]$, for $j = 1, \ldots, n$, where d is large.

Individual job weights, d large

The problem with individual weights per job, that is, $1|d_j = d|\sum_j(\alpha_j E_j + \beta_j T_j)$ is $\mathcal{NP}$-hard in the ordinary sense, even if d is large and $\alpha_j = \beta_j = w_j$, as proven by Hall and Posner (1991). They show that, as d is large, there is a job that is completed at time d, and that in the optimal schedule the early jobs are in decreasing order of w_j/p_j ratio and the tardy jobs are in increasing order of w_j/p_j ratio. Hence, if we assume the jobs to be renumbered such that

$$\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \ldots \geq \frac{w_n}{p_n},$$

then we can solve the problem by adding the jobs in the order $1, \ldots, n$, where the first job completes or starts at time d and job J_j ($j = 2, \ldots, n$) is then placed immediately in front or immediately after the contiguous subschedule of the jobs scheduled so far. The only problem is then of course: should job J_j go before or after the current subschedule? Hall and Posner solve this problem by dynamic programming. They use state variables $f_j(t)$, which denote the smallest cost of a subschedule for the jobs $1, \ldots, j$ such that the total processing time of the jobs started before time d is equal to t. The recurrence relation is then

$$f_j(t) = \min\{w_j(t - p_j) + f_{j-1}(t - p_j), w_j(\sum_{i=1}^{j} p_i - t) + f_{j-1}(t)\},$$

where the first and second term correspond to the decision of putting J_j early and tardy, respectively. This dynamic programming algorithm runs in $O(n \sum_j p_j)$ time and space.

A small adjustment of this dynamic programming algorithm solves the problem with asymmetric weights α_j and β_j, as long as these weights are agreeable, which in this case implies that the jobs can be numbered such that

$$\frac{\alpha_1}{p_1} \geq \frac{\alpha_2}{p_2} \geq \ldots \geq \frac{\alpha_n}{p_n} \quad \text{and} \quad \frac{\beta_1}{p_1} \geq \frac{\beta_2}{p_2} \geq \ldots \geq \frac{\beta_n}{p_n}.$$

If the weights are not agreeable, then the dynamic programming algorithm does not work anymore, since there is no clear order in which to add the jobs to the dynamic programming algorithm. Moreover, it is unknown whether this problem is $\mathcal{NP}$-hard in the ordinary sense or in the strong sense. Van den Akker, Hoogeveen, and Van de Velde (2002) show that this problem can be solved by formulating it as an integer linear programming problem, where the variables correspond to choosing a given subset of jobs as the early (tardy) set, or not; the only constraints needed then (except for the binary constraints) ensure that each job is included in at least one subset. The LP-relaxation is then solved by applying column generation, where surprisingly the optimal solution for the LP-relaxation is integral in each of their computational experiments (the authors provide an example to show that there can exist an integrality gap). This pure column generation approach solves instances with up to 60 jobs, but it can be improved to solve instances with up to 125 jobs by combining the column generation approach with Lagrangean relaxation. Chen and Powell (1999) apply a similar approach to minimize the same objective function in case there are m identical parallel machines. This problem can be formulated as an integer linear programming problem again, but with the additional constraint that the number of early (tardy) subschedules is at most equal to m. They use a pure column generation approach to solve the LP-relaxation, which yields a very strong lower bound, but is not integral in general. They then solve the problem to optimality by applying branch-and-bound. This algorithm can handle instances with up to 60 jobs, where $m = 2$ yields the hardest instances.

Results for a small common due date

In contrast, the problem with a small common due date is less tractable, since the optimal schedule may start at time zero rather than having a job completed at time d. Hall, Kubiak, and Sethi (1991) and Hoogeveen and Van de Velde (1991) show that the problem $1|d_j = d|\sum_j(E_j + T_j)$ with d small is $\mathcal{NP}$-hard in the ordinary sense. Hoogeveen, Oosterhout, and Van de Velde (1994) solve this problem by using branch-and-bound, where the lower bound is obtained by applying Lagrangean relaxation to the constraint that the machine is not available before time zero. This Lagrangean problem and the corresponding Lagrangean dual can be solved by the assignment approach for the large common due date problem. Next, they use the freedom in solving the assignment problem with respect to the decision of which job in each pair goes to the early part and which one goes to the tardy part to chase an optimal schedule for the Lagrangean dual problem in which the total processing time of the jobs in the early part is exactly equal to d, which is sufficient for optimality. When $n \geq 20$, then this approach virtually always finds an optimal solution in $O(n \log n)$ time, and it can be used to derive an approximation algorithm with worst-case bound 4/3. Hall, Kubiak, and Sethi (1991) propose a dynamic programming algorithm to solve the unweighted problem to optimality. This algorithm consists of two parts: one for the subcase that there is a job that is completed at time d, and one for the subcase that the first job in the schedule starts at time zero. The first subproblem can be solved by Hall and Posner's algorithm, where t is constrained to be no more than d. The second subproblem can be solved by noting that processing occurs in the interval $[0, \sum_j p_j]$ only; hence, the optimal schedule can be built using dynamic programming by starting at the borders with placing the job with largest p_j, etc. The authors show that this can be implemented to run in $O(n \sum_j p_j)$ time and space, since the job that is executed around d is the one with smallest processing time.

Finally, we come to the weighted case. Alidaee and Dragan (1997) show that the special case with $w_j = p_j$ is solved by putting the jobs in LPT-order and finding the optimal starting time, which is either zero, or such that some job is completed at time d; this algorithm can be implemented to run in $O(n \log n)$ time. For the general version of $1|d_j = d|\sum_j w_j(E_j + T_j)$ with

d small Hoogeveen and Van de Velde (1991) present a dynamic programming algorithm that runs in $O(n^2 \sum_j p_j)$ time and space; the increase in running time in comparison to the unweighted case is due to the fact that for this problem the identity of the job that is executed around d is unknown. This approach can also be used for the problem with asymmetric, agreeable weights.

5 Controllable processing times

Most classical scheduling models assume *fixed* job processing times, that is, the processing time of a job is determined a priori and cannot be modified by the scheduler. When the processing of a job requires additional resources such as manpower or funds, shorter job processing times may be accomplished by increasing these additional resources. This notion has led to the field of *scheduling with controllable processing times*, in which the processing times can be compressed at the expense of some extra cost, which is called the compression cost.

In this section, we discuss the main results, where we concentrate on the 'new' ones; we refer to the survey by Nowicki and Zdrzałka (1990) for an overview of the 'older' results. We use the notation introduced by Nowicki and Zdrzałka in their survey. The processing requirement of a job J_j is specified by three non-negative parameters a_j, u_j, and c_j, where $0 \leq u_j \leq a_j$. Here a_j is the initial processing requirement of J_j, u_j is a bound on the compressibility of J_j, and c_j models the compression cost of J_j per unit of compression. The *actual* processing time of job J_j is then equal to $p_j = a_j - x_j$, where the variable x_j with $0 \leq x_j \leq u_j$ models the amount of compression; the cost of performing this compression equals $c_j x_j$. Solving a scheduling problem with controllable processing times now amounts to (i) specifying a feasible schedule σ for the jobs together with (ii) specifying a compression vector $\vec{x}$ that encodes the compressions x_j of all the jobs. We denote the cost associated with compression vector $\vec{x}$ by

$$c(\vec{x}) = \sum_{j=1}^{n} c_j x_j.$$

We denote the scheduling cost again by $f(\sigma)$, where we assume that σ is computed on basis of the **actual** processing times. If x_j can assume any value in the interval $[0, u_j]$, then the number of Pareto optimal points is unbounded, and the best result possible is to identify the trade-off curve. If x_j can only assume integral values, then we aim at finding the set of Pareto optimal points.

Natural scheduling questions that arise are

P1: to minimize the total cost, that is, $f(\sigma) + c(\vec{x})$;

P2: to minimize $c(\vec{x})$ under the constraint $f(\sigma) \leq F$;

P3: to minimize $f(\sigma)$ under the constraint $c(\vec{x}) \leq C$;

P4: to identify the trade-off curve for $(f(\sigma), c(\vec{x}))$.

In the following, we discuss the results that have been obtained for the questions P1–P4 for a given scheduling problem, where we use the acronym 'contr' in the second field to indicate that we work with controllable processing times.

We start with the problem $1|contr|\sum_j C_j$; this was the subject of the paper Vickson (1980A) opened the area with. Vickson made the crucial observation that compressing a job J_i scheduled on position k in σ by x_i reduces $\sum_j C_j$ by $(n+1-k)x_i$ at the expense of an additional compression cost of $c_i x_i$. Hence, when $c_i \leq (n+1-k)$, then J_i should be fully compressed, and J_i should be uncompressed, otherwise. This implies that we can compute the cost c_{ik} of assigning job J_i to position k as $(n+1-k)a_i$ if $c_i \geq (n+1-k)$ and $(n+1-k)(a_i - u_i) + c_i u_i$ if $c_i \leq (n+1-k)$. Hence, problem P1 can be formulated as an assignment problem, which is solved in $O(n^{2.5})$ time. The same approach holds for P1 for the problems $P|contr|\sum_j C_j$, $Q|contr|\sum_j C_j$, and $R|contr|\sum_j C_j$ (Alidaee and Ahmadian, 1993).

Vickson (1980B) studies variant P1 of the problem $1|contr|\sum_j w_j C_j$; he describes several heuristics and a branch-and-bound algorithm to solve the problem to optimality. Vickson conjectured $\mathcal{NP}$-hardness, but this problem has stayed open for twenty years. Wan, Yen, and Li (2001) and Hoogeveen and Woeginger (2002) show that the problem is $\mathcal{NP}$-hard in the ordinary sense. The exact computational complexity is still open.

Variant P4 of the problem $1|contr|\sum_j U_j$ is studied by Cheng, Chen, and Li (1996). They provide a pseudo-polynomial dynamic programming algorithm to solve P4 under the assumption that x_j is integral; hence, by refining the unitsize, this algorithm can be used to approximate the trade-off curve in case x_j is continuous. This algorithm can also be used for the more general case in which p_j is a nondecreasing function of x_j. The authors further show that variant P2 is $\mathcal{NP}$-hard in the ordinary sense.

Vickson (1980A) considers variant P1 of the problem $1|contr|T_{\max}$ and presents an $O(n^2)$ time algorithm that solves it to optimality, if x_j can assume any value in the interval $[0, u_j]$ for all $j = 1, \ldots, n$; Vickson shows that the problem becomes $\mathcal{NP}$-hard in the ordinary sense when $x_j \in \{0, u_j\}$. A similar remark is stated by Janiak and Kovalyov (1996), who analyze problem P2 for $1|contr|T_{\max}$. They show that it is solvable in $O(n \log n)$ time, if the x_j values are allowed to be continuous; when the x_j values are assumed to be integral, then the problem is $\mathcal{NP}$-hard in the ordinary sense and solvable in pseudo-polynomial time through dynamic programming.

Van Wassenhove and Baker (1982) show that problems P1–P4 are solvable in $O(n^3)$ time for $1|contr|f_{\max}$ under the assumption that the jobs can be numbered such that

$$f_1(t) \leq f_2(t) \ldots \leq f_n(t) \quad \forall t;$$

this implies that, independent of the actual processing times, there exists an optimal schedule in which the jobs are sequenced in the order $J_1, J_2, \ldots, J_n$. Hoogeveen and Woeginger (2002) consider the special case of $1|contr|f_{\max}$ in which all penalty functions $f_j(t)$ are assumed to be piecewise linear. They show that variant P4 of this problem is solvable in time $O(L^2 n^4)$, where L denotes the number of linear pieces needed to describe all functions f_j $(j = 1, \ldots, n)$. Note that this time is polynomial if all penalty functions are described *explicitly*; the authors show that the same problem in which the penalty functions are described implicitly is $\mathcal{NP}$-hard in the strong sense.

Nowicki and Zdrzałka (1995) consider the problem with parallel machines in which preemption is allowed. They present an $O(n^2)$ greedy algorithm to solve P4 for the problem $P|contr, pmtn|C_{\max}$; in this case, the trade-off curve is piecewise linear with up to $2n+1$ breakpoints. They further show that $Q2|contr, pmtn|C_{\max}$ can be viewed upon as a special case of $P|contr, pmtn|C_{\max}$, and hence can be dealt with in the same way. For $Qm|contr, pmtn|C_{\max}$ with $m \geq 3$ they show that P2 can be solved in $O(n \max\{m, \log n\})$ time, and they use this result to approximate the trade-off curve by varying the upper bound used in P2 with ϵ per iteration and find the corresponding value of the compression cost; it is an open problem whether P4 is solvable in polynomial time for $Qm|contr, pmtn|C_{\max}$ with $m \geq 3$.

Zhang, Tang, and Chen (2001) consider P1 for the problem $R|contr|\sum_j w_jC_j$. They model this problem as a convex programming problem, and show that a relaxation of this problem formulation can be used to design a polynomial time approximation algorithm with worst-case ratio 3/2.

Nowicki and Zdrzałka (1988) consider P1 for $F2||C_{\max}$. Here, the compression of the respective operations of job J_j happens separately and independently; the cost of compressing one operation is unrelated to the compression cost of the other operation within the same job. Nowicki and Zdrzałka show this problem to be $\mathcal{NP}$-hard in the ordinary sense, even if the operations on M_2 cannot be compressed at all and all compression costs are equal. They further present an $O(n^2)$ time heuristic with worst-case bound 3/2.

Cheng and Shakhlevich (1999) study the special case of $F|contr|C_{\max}$ in which all operations belonging to the same job have equal processing time; this special case is known as the *proportionate flow shop*. In this case there exists a closed formula for the minimum makespan. Therefore, P4 is solvable in $O(n \log n)$ time under the assumption that compressing the processing time of a job implies that the processing times of all of its operations are compressed by this amount. The authors also consider variant P1 of this problem with compression costs that are no longer linear. They show that if all compression cost functions are convex, then P1 is solvable in $O(n \log n)$ time; if on the other hand all compression cost functions are concave, then P1 is solvable in $O(n^2)$ time.

Other models

So far, we have assumed that $p_j = a_j - x_j$. Shabtay and Kaspi (2003) consider the case that the actual processing time is equal to

$$p_j = \left(\frac{a_j}{x_j}\right)^k.$$

For this processing time reduction function they study P2 of the problem $1|contr|\sum_j w_jC_j$, with $c_j = 1$ $(j = 1, \dots, n)$. They show that, for a given sequence, there exists a formula to compute the optimal resource allocation $\vec{x}$. They further specify some special cases that are polynomially solvable, present a dynamic programming algorithm to find an optimal solution for the general case with running time $O(n2^n)$ and specify some heuristics.

Instead of processing times it is also possible to reduce *release dates* or *due dates* by adding additional resources. Nowicki and Zdrzałka (1990) consider the problem of minimizing the makespan on a single machine in case of controllable release dates and show that this problem, which looks quite simple, in fact is $\mathcal{NP}$-hard in the strong sense. We will indicate why this problem is hard on basis of the problem $1||L_{\max}$ with controllable due dates, which is the mirror problem of (and hence equivalent to) $1|r_j|C_{\max}$ with controllable release dates. Suppose that we have been given a sequence of jobs, and that we want to realize a certain value of $L_{\max}$, say zero, by adjusting the due dates. To achieve this we have to increase the due date d_j of J_j by $\max\{C_j - d_j, 0\}$, which equals the tardiness of job J_j in this schedule with respect to the original due dates. Hence, achieving a maximum lateness of zero boils down to the problem $1||\sum_j w_jT_j$, where w_j is equal to the cost c_j of reducing d_j. As even this simple problem is $\mathcal{NP}$-hard in the strong sense, no polynomial results are to be expected. Nowicki and Zdrzałka (1990) present no less than 10 heuristics for P1 of problem $1|r_j|C_{\max}$ with controllable release dates, but none of them is able to beat the bound of 2 with respect to the worst-case bound.

6 Other single-machine bicriteria results

In this section we will discuss results that have been obtained in single-machine scheduling that fall neither in the area of earliness-tardiness scheduling, nor in the area of scheduling with controllable processing times. We start with lexicographical optimization.

Lexicografical optimization

If the more important criterion is total completion time, then the only freedom left for minimizing the second criterion is to reschedule jobs with equal processing time, which is solvable in polynomial time for all standard criteria including total weighted tardiness. Hence, all lexicographical optimization problems with $\sum_j C_j$ as the primary criterion are solvable in polynomial time. A similar observation can be made for the lexicographical problem with total weighted completion time as the more important criterion: the only freedom left is to reschedule the jobs with equal $\frac{w_j}{p_j}$ ratio, and since this does not influence the secondary problem, we find that the lexicographical problem with $\sum_j w_j C_j$ as the primary criterion is as hard as the problem of minimizing just the secondary criterion.

Now let the primary criterion be of the minimax type, like $L_{\max}$, $T_{\max}$, or $f_{\max}$. Optimality with respect to the primary criterion yields a set of deadlines that have to be obeyed when minimizing the secondary criterion. The resulting problem is polynomially solvable when the secondary criterion is one of $\sum_j C_j$, $L_{\max}$, $T_{\max}$, or $f_{\max}$ and strongly $\mathcal{NP}$-hard for $\sum_j w_j C_j$. The problem with $E_{\max}$ is strongly $\mathcal{NP}$-hard if the primary criterion is $f_{\max}$ and polynomially solvable if the primary criterion is $L_{\max}$ or $T_{\max}$, which difference in behavior is due to the relation between the due dates and the deadlines in the latter case. If the secondary criterion is to minimize the number of late jobs, $\sum_j U_j$, then the problem is $\mathcal{NP}$-hard in the ordinary sense for primary criterion $f_{\max}$ (Lawler, 1982) and open for primary criteria $L_{\max}$ and $T_{\max}$.

The last case that we consider is the class of hierarchical minimization problems that have $\sum_j U_j$ as the primary criterion. The computational complexity for the secondary criteria of $L_{\max}$, $T_{\max}$, and $E_{\max}$ is still open. The $1||Lex(\sum_j U_j, T_{\max})$ problem has been attacked by Shanthikumar (1983) using branch-and-bound; Gupta, Hariri, and Potts (1999) present an improved branch-and-bound algorithm that is based on a new lower bounding scheme. The $1||Lex(\sum_j U_j, f_{\max})$ problem can be shown to be $\mathcal{NP}$-hard through the $\mathcal{NP}$-hardness proof by Lawler (1982) for $1|\bar{d}_j|\sum_j U_j$. Similarly, the $\mathcal{NP}$-hardness proof by Du and Leung (1990) showing ordinary $\mathcal{NP}$-hardness for $1||\sum_j T_j$ can be adjusted to show that $1||Lex(\sum_j U_j, \sum_j T_j)$ is at least $\mathcal{NP}$-hard in the ordinary sense; this problem has been investigated by Vairaktarakis and Lee(1995), who present a branch-and-bound algorithm for it. Chen and Bulfin (1993) show that the problem $1||Lex(\sum_j U_j, \sum_j C_j)$ is $\mathcal{NP}$-hard using a so-called id-encoding. This result is weaker than ordinary $\mathcal{NP}$-hardness, and an $\mathcal{NP}$-hardness proof using the standard encoding would be a welcome result. Emmons (1975A) presents a branch-and-bound algorithm for this problem. The strong $\mathcal{NP}$-hardness of the problem $1||Lex(\sum_j U_j, \sum_j w_j C_j)$ follows immediately from the strong $\mathcal{NP}$-hardness of $1|\bar{d}_j|\sum_j w_j C_j$, where the deadlines are introduced by requiring $\sum_j U_j = 0$.

Chen and Bulfin (1990) concentrate on problems with unit processing times, that is, $p_j = 1$ for all jobs J_j $(j = 1, \ldots, n)$. They consider combinations of the performance criteria $T_{\max}$, $\sum_j w_j T_j$, $\sum_j w_j C_j$, and $\sum_j w_j U_j$. Because of the unit processing times, each one of the lexicographical problems is solvable by solving at most two assignment problems, albeit that for many combinations faster algorithms are available.

Minimizing a linear composite objective function

Here we only mention results that are specific to a linear composite objective function and therefore do not hold for a general one. Remark that we denote this problem by $1||F_l(f,g)$; the outcome value for a given schedule σ is then equal to $f(\sigma)+\alpha g(\sigma)$. The first problem for which it is easy to minimize a linear composite objective function and hard to minimize a general composite objective function is to minimize a combination of two total weighted completion time criteria with different sets of weights, that is, $\sum_j w_j C_j$ and $\sum_j \hat{w}_j C_j$. Given α, which is the case in a priori optimization, the objective function $\sum_j w_j C_j + \alpha \sum_j \hat{w}_j C_j$ can be rewritten as $\sum_j \bar{w}_j C_j$, where $\bar{w}_j = w_j + \alpha \hat{w}_j$. Hence, we can solve this problem by putting the jobs in order of non-increasing $\bar{w}_j/p_j$-ratio. If on the contrary we deal with a posteriori optimization, in which α is not given, then we can construct the efficient frontier using parametric analysis as described in Subsection 3.2. Thereto, we compute for each pair of jobs J_i and J_j for which $\frac{w_i}{p_i} > \frac{w_j}{p_j}$ and $\frac{\hat{w}_i}{\hat{p}_i} < \frac{\hat{w}_j}{\hat{p}_j}$ the value of α_{ij} such that

$$\frac{w_i + \alpha_{ij}\hat{w}_i}{p_i} = \frac{w_j + \alpha_{ij}\hat{w}_j}{p_j};$$

when $\alpha < \alpha_{ij}$, then job J_i precedes job J_j in the optimum schedule, whereas for values $\alpha > \alpha_{ij}$ the opposite holds. Hence, if we order the α_{ij} values and compute the corresponding schedules, then we find the extreme schedules, since the optimum schedule for any value of α between two consecutive α_{ij} values is equal to the schedule corresponding to the smaller of these two α_{ij} values. Finding the set of extreme schedules can be done in $O(n^2 \log n)$ time, since we only need to sort the $O(n^2)$ α_{ij} values, compute the optimum schedule for $1||Lex(\sum_j w_j C_j, \sum_j \hat{w}_j C_j)$, after which we only have to interchange two jobs to find the next extreme schedule in each iteration. Note that the efficient frontier is not equal to the trade-off curve in this case; Lenstra (-; see Hoogeveen (1992)) showed that the finding a specific Pareto optimal point is at least $\mathcal{NP}$-hard in this case by showing that the problem $1|\sum_j \hat{w} C_j \le W|\sum_j w_j C_j$ is $\mathcal{NP}$-hard in the ordinary sense. A similar approach has been used by Bagchi (1989) for minimizing a nondecreasing linear function of $\sum_j C_j$ and $\sum_i \sum_j |C_j - C_i|$. Given α the objective function can be reformulated in terms of positional weights, which makes it possible to compute the $(n-1)$ values α explicitly at which the schedule that minimizes the objective function changes. Bagchi (1989) shows that the corresponding schedules can be determined in $O(n^2)$ time.

In the problem described above it is straightforward to solve the a priori problem, since we can compute an optimum solution for a given value of α in $O(n \log n)$ time; we do not need the set of extreme schedules then, and hence the a priori problem is easier than the a posteriori problem. In the case of the simultaneous minimization of $P_{\max}$ and $\sum_j C_j$, however, we need to construct the efficient frontier to solve the a priori problem, which is hence as difficult as the a priori problem. The problem of minimizing $\sum_j C_j$ subject to an upper bound on $P_{\max}$ is $\mathcal{NP}$-hard in the strong sense, but it becomes polynomially solvable when preemption is allowed; note that we use $P_{\max}$ instead of $E_{\max}$, since the latter makes no sense in case of preemption. Because of the possibility of preempting at any point in time, the number of Pareto optimal points is infinite, but Hoogeveen and Van de Velde (2001) show that it is possible to find the efficient frontier in $O(n^4)$ time, as there are at most $O(n^2)$ extreme points, which can be determined in $O(n^2)$ time per point. Note that the trade-off curve is not convex in this case. They further show that the optimal schedule will be nonpreemptive if the total completion time criterion outweighs maximum promptness.

Chen and Bulfin (1990) show that for the unit processing time case the problem of minimizing a linear composite objective function of two criteria of the sum type, like $\sum_j w_j C_j$, $\sum_j w_j U_j$, or $\sum_j w_j T_j$, is solvable as an assignment problem when α is given. Since we can solve any assignment problem as a linear programming problem, we can find the set of extreme schedules by determining the efficient frontier of the linear programming relaxation through parametric analysis, just like the problem of producing glass bottles described in Subsection 3.2.

Minimizing a general composite objective function

The $1||F(L_{\max}, \sum_j C_j)$ problem has already been discussed in Subsection 3.1. Hoogeveen and Van de Velde (1995) show that exactly the same approach can be used to solve the $1||F(\sum_j C_j, f_{\max})$ problem, where in each iteration the subproblem $1|f_{\max} < f|\sum_j C_j$ is solved, with f equal to the $f_{\max}$ value of the previous Pareto optimal point. Emmons (1975B) shows that this subproblem can be solved in $O(n^2)$ time. An alternative is to compute the deadlines induced by the constraint $f_{\max} < f$, which can be achieved using bisection if the function f_j does not have an inverse and then use Smith's rule; this takes $O(n(\log n + \log(\sum_j p_j)))$ time. Therefore, the problem can be solved in $O(n^3 \min\{n, \log n + \log(\sum_j p_j)\})$ time.

Hoogeveen (1996B) shows that the problem of minimizing two maximum cost criteria, that is $1||F(f_{\max}, g_{\max})$, is solvable in $O(n^4)$ time. This algorithm is based on the ϵ-constraint approach: there are $O(n^2)$ Pareto optimal points and each subproblem is solved in $O(n^2)$ time. When the maximum cost criterion $g_{\max}$ is such that (possibly after renumbering) $g_1(t) \leq g_2(t) \leq \ldots \leq g_n(t)$ for all $t \in [0, \sum_j p_j]$, then the running time can be improved to $O(n^3 \log n)$; note that $L_{\max}$ satisfies this constraint. Hoogeveen further shows that the problem of simultaneously minimizing three maximum cost criteria is solvable in $O(n^8)$ time by a generalization of the above algorithm; here it takes $O(n^2)$ time to find one Pareto optimal point, the number of which is bounded by $O(n^6)$. This algorithm can be generalized even further to find all Pareto optimal points for a combination of K maximum cost criteria, but since there is no polynomial upper bound known on the number of Pareto optimal points for $K \geq 4$, it is unclear whether this algorithm is polynomial for fixed K (for arbitrary K the problem is strongly $\mathcal{NP}$-hard). Sourd (2001) considers the problem $1|r_j, pmtn|F(f_{\max}, g_{\max})$. He shows that the number of Pareto optimal points is again bounded by $O(n^2)$ and that each Pareto optimal point can be determined in $O(n^2)$ time, which leads to an $O(n^4)$ algorithm altogether.

The only other problems that are polynomially solvable, as far as we know, are problems with unit processing times. Chen and Bulfin (1990) show that then the set of Pareto optimal points can be determined by applying the ϵ-constraint approach for any combination of two performance criteria, one of which is $T_{\max}$, since each of the subproblems can then be solved as an assignment problem. We claim that a similar result can be achieved with $f_{\max}$ or $E_{\max}$ instead of $T_{\max}$.

The remaining problems are $\mathcal{NP}$-hard or open. Many researchers have proposed branch-and-bound algorithms or heuristics for these problems; we will mention only the most notable ones. Nelson, Sarin, and Daniels (1986) study the problems $1||F(\sum_j C_j, \sum_j U_j)$ and $1||F(T_{\max}, \sum_j U_j)$. They attack each of the problems through the ϵ-constraint approach with an upper bound on $\sum_j U_j$; these subproblems are solved by applying branch-and-bound. Karasakal and Köksalan (2000) attack the $1||F(\sum_j C_j, \sum_j U_j)$ problem in the same way, but instead of solving the subproblem to optimality they use simulated annealing. Tadei, Grosso, and Della Croce (2002) apply the ϵ-constraint approach to the problem $1||F(T_{\max}, \sum_j T_j)$; in each step they solve a problem of the form $1|T_{\max} \leq T|\sum_j T_j$ using branch-and-bound. Köksalan, Azizoglu, and Kondakci (1998) present a heuristic to determine the trade-off curve for the single-machine problem with criteria $E_{\max}$ and $\sum_j C_j$ through the ϵ-constraint method; these results have been improved by Karasakal and Köksalan (2000) through simulated annealing.

7 Results for multiple machines

In this section we consider models in which more than one machine is available for processing the jobs. We distinguish between problems with parallel machines and shop models. In case of parallel machines, each job has to be processed by just one of the available machines. Therefore, these problems consist of an assignment phase, in which we pick the machine that will execute the job, and a sequencing phase, in which we determine the order in which each machine is going to execute the jobs that have been assigned to it. In case of a shop model each machine executes one of the operations that the job consists of. We first deal with the parallel machine problems; the shop models are discussed in the second subsection.

7.1 Parallel machine scheduling

It is well-known that the problems $P||\sum_j C_j$ and $Q||\sum_j C_j$ are solved in $O(n \log n)$ time through the technique of the positional weights (see Subsection 4.2), whereas the $R||\sum_j C_j$ is solvable as an assignment problem (Horn, 1973). Unfortunately, $\sum_j C_j$ is the only criterion that can be minimized in polynomial time, as the problem $P2||C_{\max}$ is already $\mathcal{NP}$-hard (in the ordinary sense). Moreover, the problem $P2||Lex(\sum_j C_j, C_{\max})$ can easily be shown to be $\mathcal{NP}$-hard in the ordinary sense through a reduction from Even-Odd Partition; for this problem, Eck and Pinedo (1993) present an $O(n \log n)$ algorithm that finds a solution with makespan no more than $\frac{28}{27}$ times the minimum makespan given that the total completion time is minimum.

Although there is no bicriteria scheduling problem with general processing times that can be solved to optimality in polynomial time, the situation becomes much better when preemption is allowed or when all processing times are assumed to be identical and the machines are either identical or uniform. We start with the latter case. It is well-known that, in case of a regular performance criterion, it is optimal to consider only the sequence of n earliest completion times, and then to assign the jobs to these completion times optimally. This requires $O(n^{2.5})$ time in the worst case, but for many criteria the assignment problem can be solved by a faster algorithm; we refer to Dessouky, Lageweg, Lenstra, and Van de Velde (1990) for details. Since an upper bound on a minimax criterion like $L_{\max}$ or $f_{\max}$ can easily be incorporated in the assignment problem, the Pareto optimal set can be determined for any combination of a sum criterion and a minimax criterion or for two minimax criteria by applying the ϵ-constraint approach. The running time of the algorithm depends on the number of Pareto optimal points that we discover. Tuzikov, Makhaniok, and Männer (1998) show that in both variants there are at most $O(n^2)$ Pareto optimal points. The problem of minimizing a *linear* function of two sum criteria with a given relative weight α is solvable as an assignment problem, since we can express the cost c_{ij} of assigning job J_j $(j = 1, \ldots, n)$ to the ith smallest completion time $C_{[i]}$ $(i = 1, \ldots, n)$ as $c_{ij} = d_{ij} + \alpha e_{ij}$, where d_{ij} and e_{ij} denote the cost component of the first and second component respectively. Since we can solve an assignment problem as a linear programming problem, we can find the set of extreme schedules by determining the efficient frontier through parametric analysis, just like the problem of producing glass bottles described in Subsection 3.2. Note that the efficient frontier is not equal to the trade-off curve in this case; the problem with a general composite objective function is open in this case.

We continue with the case in which preemption is allowed. Note that then the number of Pareto optimal points is likely to be unbounded. Therefore, we aim at finding the efficient frontier and possibly the trade-off curve. We first discuss the combination $C_{\max}$ and $L_{\max}$. Since Sahni (1979) showed that the problem $P|\bar{d}_j, pmtn|-$, where the $-$ in the third field indicates that it is a feasibility problem, is solvable in $O(n \max\{m, \log n\})$ time, we can apply the ϵ-constraint method where we put an upper bound on $L_{\max}$. The main problem is of course to find the upper bound values that correspond to an extreme point. Mohri, Masuda, and Ishii (1999) show that for the two-machine case the trade-off curve is the line segment connecting the points corresponding to the two lexicographical optimization problems $P2|pmtn|Lex(C_{\max}, L_{\max})$ and $P3|pmtn|Lex(L_{\max}, C_{\max})$, which points may coincide; they further give an explicit mathematical expression of these two points. Mohri et al. present a partial description of the trade-off curve for the three-machine case as well. T'kindt and Billaut (2002) present a full description of the trade-off curve for this problem: it consists of at most two line-segments. As far as we know, no results have been reported for $m > 3$.

We continue with the combination of $C_{\max}$ and $\sum_j C_j$. Leung and Young (1989) show that $P|pmtn|Lex(\sum_j C_j, C_{\max})$ is solvable in $O(n \log n)$ time. McCormick and Pinedo (1995) study the $Q|pmtn|F(\sum_j C_j, C_{\max})$ problem. They apply the ϵ-constraint method, where an upper bound is put on $C_{\max}$, and show that this subproblem is minimized in $O(nm)$ time. Moreover, they show that they can determine the efficient frontier, which is equal to the trade-off curve in this case, in $O(nm^3)$ time altogether, where the number of breakpoints is $O(nm)$.

7.2 Shop models

The flow shop

We start with the flow shop model. Since only $F2||C_{\max}$ is solvable in polynomial time in case of general processing times, there are no bicriteria problems that are solvable in polynomial time. Therefore, we concentrate on heuristics and enumerative approaches. The overwhelming majority of research on bicriteria flow shop problems concerns the two-machine case in which some combination of $\sum_j C_j$ and $C_{\max}$ is to be minimized. The $F2||Lex(C_{\max}, \sum_j C_j)$ is already $\mathcal{NP}$-hard in the strong sense, which follows from the proof that $F2||\sum_j C_j$ is $\mathcal{NP}$-hard in the strong sense (Garey, Johnson, and Sethi, 1976). Heuristics for this problem have been presented by Rajendran (1992), Gupta, Neppalli, and Werner (2001), and T'kindt, Gupta, and Billaut (2003). Local search algorithms have been proposed by T'kindt, Monmarché, Tercinet, and Laügt (2002), who apply the technique of ant colony optimization, and Huang and Lim (2003), who apply a technique that they call *Local Dynamic Programming*, which tries to improve a solution by replacing a small part of the current solution by an optimal solution for that part that has been found by applying dynamic programming; the latter seems to work best. The currently best branch-and-bound algorithm, which can solve problem instance of up to 35 jobs to optimality, is due to T'kindt, Gupta, and Billaut (2003). The $F2||F_l(C_{\max}, \sum_j C_j)$ problem has been studied by Nagar, Heragu, and Haddock (1995) and Sivrikaya-Serifoglu and Ulusoy (1998). In both papers heuristics and branch-and-bound algorithms are presented, which are capable of solving instances with no more than a disappointing 18 jobs. Finally, Sayin and Karabati (1999) solve the $F2||F(C_{\max}, \sum_j C_j)$ problem by generating the Pareto optimal set through the ϵ-constraint approach. They apply branch-and-bound to solve the subproblems, which are derived by putting an upper bound on the makespan. Their computational experiments show that only seldomly there exists a Pareto optimal point that is not found when applying lexicographical optimization.

The $F2||F(C_{\max}, T_{\max})$ has been studied by Daniels and Chambers (1990), who present a branch-and-bound algorithm for it and a heuristic to approximate the set. In general, the cardinality of the Pareto optimal set is no more than 3. They also present a heuristic to approximate the Pareto optimal set for the m-machine case, where they restrict theirselves to permutation schedules (which are optimal for $m = 2$, but not for $m \geq 3$).

Liao, Yu, and Joe (1997) study the problems $F2||F(C_{\max}, \sum_j U_j)$ and $F2||F(C_{\max}, \sum_j T_j)$. For both problems they present a branch-and-bound algorithm that solves instances with up to 30 jobs. The number of Pareto optimal points is usually at most 2 in the first case and at most 3 or 4 in the second case.

The open shop

Now we consider the open shop problem. Again, the two-machine hierarchical problem of minimizing $\sum_j C_j$ subject to minimum makespan has attracted most attention, especially since there exists an easy formula for the minimum makespan (Gonzalez and Sahni, 1976). Just like the flow shop problem, it follows immediately from the strong $\mathcal{NP}$-hardness of the $O2|| \sum_j C_j$ problem (Achugbue and Chin, 1982) that $O2||Lex(C_{\max}, \sum_j C_j)$ problem is $\mathcal{NP}$-hard in the strong sense. Kyparisis and Koulamas (2000) identify three special cases that are polynomially solvable. The two most important ones are that the makespan is determined by the longest job, and that the minimum processing time of the jobs on machine M_1 is at least twice the maximum processing time of the jobs on machine M_2. They show that, if the second criterion holds for the maximum processing time on M_2 and M_3, then the three-machine case is polynomially solvable as well. The authors further present some heuristics for the two and three-machine case under the assumption that one machine is *dominant*, that is, the minimum processing time on this machine is at least equal to the maximum processing time on the other machines, and analyze the worst-case performance behavior of these heuristics.

Masuda and Ishii (1994) study the $O2|pmtn|F(C_{\max}, L_{\max})$ problem. They show that the trade-off curve is the line segment connecting the points corresponding to the two lexicographical optimization problems $O2|pmtn|Lex(C_{\max}, L_{\max})$ and $O2|pmtn|Lex(L_{\max}, C_{\max})$, which points may coincide. These two points can easily be computed using the $O(n \log n)$ algorithm by Lawler, Lenstra, and Rinnooy Kan (1981) for the $O2|\bar{d}_j, pmtn|-$ problem and the expression for the outcome value of $O2|pmtn|L_{\max}$ by the same authors. Note the striking resemblance to the results obtained for the similar problem with two parallel, identical machines.

8 Simultaneous bicriteria approximation

In this section we discuss results that have been obtained in the area of bicriteria worst-case analysis. Given two criteria f and g, the question is to find (or to show the existence of) a schedule σ such that

$$\frac{f(\sigma)}{f^*} \leq \alpha \quad \text{and} \quad \frac{g(\sigma)}{g^*} \leq \beta,$$

where α and β are two given numbers; such a schedule is then called an (α, β)-schedule. The optimum result here would be to find the curve that gives for each value α the minimum value β such that the combination (α, β) is feasible. Note that knowledge of the trade-off curve is not sufficient, since the bounds must hold for any instance.

The first result in this area has been obtained by combining the results by Graham (1966) and Kawaguchi and Kyan (1986) for the problem of scheduling jobs on m identical, parallel machines. Graham showed that list scheduling, which is a greedy heuristic in which always the first unscheduled job in the list is assigned to the first machine that becomes available, is a $(2-1/m)$-approximation algorithm when minimizing the makespan, irrespective of the way the jobs were put in the list, and Kawaguchi and Kyan showed that list scheduling is a $(\sqrt{2}+1)/2$-approximation algorithm for the problem of minimizing total weighted completion time, when the jobs are put in the list in *SWPT*-order, that is, in order of non-increasing w_j/p_j-ratio.

A second result on simultaneous approximation, which is a negative result, has been derived, also by coincidence, for the problem of minimizing the makespan and total completion time on a set of parallel unrelated machines. For the $R||C_{\max}$ a 2-approximation algorithm exists, which is due to Lenstra, Tardos, and Shmoys (1990). It was conjectured that the optimal schedule for the $R||\sum_j C_j$ problem, which can be determined in $O(n^{2.5})$ time by solving an assignment problem (Horn, 1973), is a 2-approximation algorithm for $R||C_{\max}$, too. Hurkens and Coster (-) show by counterexample that there exist instances for which the optimal solution for $R||\sum_j C_j$ can have a makespan that is $\Omega(\log n)$ times the optimum makespan.

Chakrabarti, Phillips, Schulz, Shmoys, Stein, and Wein (1996) are the first, as far as we know, to recognize the simultaneous approximation problem as a separate research area. They describe a procedure that constructs a $(2,2)$ schedule for minimizing makespan and total weighted completion time, *given an optimal solution to the single-criteria problems*. This approach has been described in more detail in a paper by Stein and Wein (1997), who refined the analysis to obtain some better bounds. We describe their basic procedure below; it is valid under some mild conditions, which are satisfied by almost all scheduling environments. These conditions are

1. *Truncation at time t*: Let σ be a valid schedule, and let $\bar{\sigma}$ be the schedule that remains when all jobs that are completed after time t are removed from σ. Then $\bar{\sigma}$ is valid for the remaining jobs.

2. *Composition*: Let σ_1 and σ_2 be two valid schedules for job sets $\mathcal{J}_1$ and $\mathcal{J}_2$, respectively. Then the schedule σ that is composed by appending σ_2 to the end of σ_1 and removing the jobs in $\mathcal{J}_1 \cap \mathcal{J}_2$ from σ_2 is a valid schedule for the jobs in $\mathcal{J}_1 \cup \mathcal{J}_2$.

The procedure now works as follows. Suppose we are given an optimum schedule π_1 for the problem of minimizing total weighted completion time and an optimum schedule π_2 for the problem of minimizing the makespan. Construct schedule σ_1 from π_1 by truncation at time $C_{\max}(\pi_2) = C^*_{\max}$. Schedule σ_2 is constructed from schedule π_2 by removing all jobs that appear in σ_1 without changing the starting times of the remaining jobs. Finally σ is constructed by composition of σ_1 and σ_2.

Since the last job in both σ_1 and σ_2 is completed by time $C^*_{\max}$, we find that $C_{\max}(\sigma) \le 2C^*_{\max}$. Moreover, the completion time of any job J_j that appears in σ_1 is equal to its completion time in π_1, whereas any job J_j in σ_2 is completed by time $2C^*_{\max}$, which is less than twice its completion time in π_1, in which it was completed after time $C^*_{\max}$. Hence, the total weighted completion time of σ is at most twice the optimal total weighted completion time. Therefore, the resulting schedule is a $(2,2)$-schedule for the criteria of makespan and total weighted completion time. Note that this result only shows the *existence* of such a schedule. If the optimal schedules for the single-criterion problems are not known, then we can use an approximation. In general, if we have approximation algorithms with worst-case bound λ and μ for the problem of minimizing the makespan and total weighted completion time, respectively, and we apply the above approach to these two schedules, then we find a $(2\lambda, 2\mu)$-schedule.

By varying the truncation point, (α, β)-schedules can be constructed for other values (α, β). Stein and Wein show how to find, for example, a $(2, 1.735)$-schedule, a $(1.785, 2)$-schedule, and a $(1.88, 1.88)$-schedule. At the negative side, they show that it is not possible to find an (α, β)-schedule with both $\alpha \leq (\sqrt{5}+1)/2$ and $\beta \leq (\sqrt{5}+1)/2$.

These results have been improved by Aslam, Rasala, Stein, and Young (1999). Aslam et al. use the procedure by Stein and Wein, but consider all possible truncation points. They show that for any $\rho \in [0, 1]$ there exists an $(1+\rho, e^\rho/(e^\rho - 1))$-schedule.

Torng and Uthaisombut (1999) show that for the specific environment of a single-machine with release dates the bound above is tight. They present an algorithm that for each $\rho \in [0, 1]$ finds an $(1+\rho, e^\rho/(e^\rho - 1))$-schedule, and they show that there exist instances for this problem for which no (α, β)-schedule for $C_{\max}$ and $\sum_j C_j$ exists with $\alpha = 1+\rho$ and $\beta < e^\rho/(e^\rho - 1))$ for any $\rho \in [0, 1]$.

Rasala (1999) extends the analysis to other combinations of performance criteria, where one of these is a minimax criterion and the other one is a sum criterion. The first extension is to consider maximum lateness and total weighted lateness. To avoid the situation that the optimum value is zero or even negative, in which case a worst-case bound of an approximation algorithm does not make sense, the common assumption of nonnegative due dates is made. But in this case the maximum lateness and the total weighted lateness criterion show the same behavior as the maximum completion time and the total weighted completion time criterion. Therefore, replacing $C_{\max}$ by $L_{\max}$ and/or $\sum_j w_j C_j$ by $\sum_j w_j L_j$ yields the same result again; this also holds for the results described below.

The second extension concerns flow time, where the flow time of job J_j is defined as $F_j = C_j - r_j$, which models the time that job J_j stays in the system. For the combination of $C_{\max}$ and $\sum_j w_j F_j$, there exists a $(2+\rho, e^\rho/(e^\rho - 1))$-schedule for any $\rho \in [0, 1]$. For the combination of $F_{\max}$ and $\sum_j w_j C_j$, there exists a $(1+\rho, 2+1/\rho)$-schedule for any $\rho \in [0, 1]$. On the contrary, for the combination $F_{\max}$ and $\sum_j F_j$ it is not possible to find an (α, β)-schedule with α and β simultaneously smaller than $\sqrt{n}/4$.

The third extension concerns maximizing the number of on-time jobs, which is denoted as $\sum_j (1 - U_j)$. Although this is a maximization problem, we define a λ-approximation algorithm as an algorithm that produces a solution that is at most a factor λ worse than an optimal solution. It can be easily shown that there exists a $(2, 2)$-schedule for $C_{\max}$ and $\sum_j (1 - U_j)$. Moreover, this bound is tight for the single-machine problem with release dates. Furthermore, a nonexistence result is shown for the problem involving $F_{\max}$ and $\sum_j (1 - U_j)$, which states that it is not possible to approximate both within a factor of $n^{\frac{1}{4}}$ simultaneously. Note that the combination of $L_{\max}$ and $\sum_j (1 - U_j)$ only makes sense if both criteria use a different set of due dates.

9 New developments

In this section we consider new bicriteria models that have recently appeared or are to appear in the literature. In the first subsection we look at the problem with interfering job sets, that is, there are two job sets, which compete for processing on a single machine. In the second subsection we assume that it is possible to reject jobs at a certain price; this rejection cost has then to be added to the scheduling cost. In the final subsection we consider the problem in which an existing schedule has to be accommodated to deal with a set of new jobs; here, we have the obvious conflict between the goals of finding an optimal solution for the entire instance, while staying as close to the original solution as possible.

9.1 Interfering job sets

In the description of bicriteria scheduling so far, we have assumed that there was one set of jobs that were evaluated by two performance criteria. Recently, Baker and Smith (2003) proposed a model in which *two sets of jobs* have to be executed by a single machine at the same time, where each set of jobs is evaluated with respect to its own performance criterion. The quality of the solution is then computed as the sum of the first outcome value plus θ times the second outcome values, where $\theta > 0$. The authors consider the performance criteria $C_{\max}$, $L_{\max}$, $\sum_j C_j$, and $\sum_j w_j C_j$. They show that if job set 1 is evaluated on basis of $C_{\max}$, then there exists an optimal solution in which these jobs are processed contiguously, and hence they can be replaced by a single job. Therefore, $C_{\max}$ in combination with any other criterion is solvable in a straightforward manner. If job set 1 has performance criterion either $L_{\max}$ or $\sum_j C_j$, then there exists an optimal solution in which the jobs are in EDD-order and SPT-order, respectively. Therefore, combining $L_{\max}$ with $L_{\max}$ or $\sum_j C_j$ can be solved by finding the optimal interleaving of these two sequences, which can be done in polynomial time by using dynamic programming. Anyone of the four combinations involving $\sum_j C_j$ and $\sum_j w_j C_j$ can be solved by merging the two job sets and executing them in $SWPT$-order, where the weights of the jobs in the second set are multiplied by θ. The only remaining combination is $\sum_j w_j C_j$ and $L_{\max}$. In this case the $SWPT$-order is only optimal for the corresponding set of jobs if it coincides with the SPT-order. Consequently, this problem can be shown to be $\mathcal{NP}$-hard in the strong sense.

9.2 Scheduling with rejection

One of the main assumptions in scheduling is that each existing job must be executed, which is a logical statement for the work floor. But on a higher level in the organization the decision can be made to reject jobs that do not fit in the current work plan; the cost incurred with this is called the *rejection cost*. Naturally, this cost should be added to the cost of the schedule, and this total cost must then be minimized. So far, most of the research has concentrated on approximation algorithms in an on-line or off-line setting.

The first two papers in this area are due to Bartal, Leonardi, Marchetti-Spaccamela, Sgall, and Stougie (2000) and Engels, Karger, Kolliopoulos, Sengupta, Uma, and Wein (2003). Engels et al. consider the problem of minimizing the sum of the total weighted completion time and total rejection cost. They show that the single-machine problem is already $\mathcal{NP}$-hard in the ordinary sense, but that it can be solved to optimality using dynamic programming. They further show how the concept of rejecting jobs can be incorporated in a linear programming framework that can be applied to derive approximation algorithms with a constant worst-case bound.

Bartal et al. consider the problem of minimizing the makespan plus the rejection cost on m parallel identical machines. For the on-line problem (that is, no information is known about the future) they present an algorithm with worst-case ratio 2.618 for an arbitrary number of machines, and they show that this result is best possible. They further show that for $m = 2$ and $m = 3$ worst-case ratios of 1.618 and 2 can be achieved, respectively, where the first one is best possible. The authors also consider this problem in an off-line setting (that is, all information about the instance is available at time zero). They present a fully polynomial approximation scheme for fixed m, a polynomial approximation scheme for the case that m is part of the input, and an $O(n \log n)$ time algorithm with worst-case bound $2 - \frac{1}{m}$.

Hoogeveen, Skutella, and Woeginger (2003) also consider the problem of minimizing the makespan plus the rejection penalty, but they allow preemption. They show that this problem is already $\mathcal{NP}$-hard in the ordinary sense when there are two parallel identical machines, but that for any number of parallel identical or uniform machines and for any fixed number of unrelated parallel machines the problem is solvable in pseudo-polynomial time and that then there exists a fully polynomial approximation scheme. When the number of unrelated parallel machines is part of the input, the problem becomes strongly $\mathcal{NP}$-hard and APX-hard (which implies that there cannot exist a polynomial approximation scheme unless $\mathcal{P} = \mathcal{NP}$), even if all rejection penalties are equal; the authors present a 1.58-approximation algorithm for this case. Finally, the authors show that when this problem is put in the open shop environment, then similar results can be obtained as for the unrelated parallel machines case.

9.3 Rescheduling for new orders

In many situations issuing a schedule to the workfloor initiates the preparations needed to execute the jobs smoothly according to plan, even though the job itself has not started yet. Therefore, when disturbances cause the need for a new plan, it would be nice to keep the new plan as close to the original plan as possible. On the other hand, the quality of the new plan should be as high as possible. These conflicting interests lie on the basis of the single-machine bicriteria scheduling problem studied by Hall and Potts (2003).

The main part of the paper by Hall and Potts is devoted to the situation in which for the current set of jobs, which we call the old jobs from now on, an optimal schedule has been determined, after which a new set of jobs (the so-called new jobs) come in, and a new schedule must be determined. Given this new schedule, the *disruption cost* is then computed for the old jobs on basis of the difference between the old and the new schedule. Hall and Potts consider two measures to compute the diffence for any old job J_j. In the first case, this difference is computed as the difference between the position of job J_j in the new schedule and its position in the old schedule; this is denoted by $D_j(\pi^*, \sigma)$, where π^* denotes the old schedule (where the $*$ indicates its optimality) and σ denotes the new schedule. In the second case, the cost for job J_j is measured by the difference in completion time; this is denoted by $\Delta_j(\pi^*, \sigma)$. For each of these cost functions, the overall disruption cost is then computed as, either the sum, or the maximum over the old jobs. Hence, there are four ways to compute the disruption cost:

$$D_{\max}(\sigma) = \max_j D_j(\pi^*, \sigma); \quad \sum_j D_j(\sigma) = \sum_j D_j(\pi^*, \sigma);$$

$$\Delta_{\max}(\sigma) = \max_j \Delta_j(\pi^*, \sigma); \quad \text{and} \sum_j \Delta_j(\sigma) = \sum_j \Delta_j(\pi^*, \sigma).$$

Hall and Potts consider two criteria to measure the quality of the plan, which are maximum lateness and total completion time; this scheduling cost is computed on basis of all jobs. For each combination of a disruption cost and a scheduling cost, Hall and Potts consider the *constrained* version and the *total cost* version of the problem. In the constrained version of the problem the scheduling cost has to be minimized subject to an upper bound on the disruption cost, whereas in the total cost version the goal is to minimize a linear function of the scheduling cost and the disruption cost. The authors have looked at all of these 16 combinations. For eleven of them the authors determine a polynomial time algorithm, two are proven $\mathcal{NP}$-hard in the strong sense, whereas for one problem the authors present a pseudo-polynomial algorithm together with a proof that the problem is $\mathcal{NP}$-hard in the ordinary sense; two problems are still open.

The key to finding the polynomial algorithms is formed by the observation that all of the above problems in which the scheduling cost is equal to the total completion time there exists an optimal solution in which all old jobs are in SPT-order and all new jobs are in SPT-order (but an old job and a new job do not have to be in SPT-order); such a schedule is called an (SPT,SPT)-schedule. Moreover, a similar result exists for the variants in which the scheduling cost is measured by the maximum lateness and the disruption cost is either $D_{\max}$ or $\Delta_{\max}$: here all new jobs are in EDD-order and all old jobs are in EDD-order; such a schedule is called an (EDD,EDD)-schedule. If the separate orderings of the old and new jobs are known, then the only problem left is to find the optimal way to interleave these two sequences, which is often determined by applying dynamic programming.

Hall and Potts further consider the problem in which a series of disruptions take place. The difference with the case of having a single disruption is that after the first disruption the incumbent schedule does not have to be optimal anymore for the whole set of jobs with respect to the scheduling cost. This complicates the problem severely.

Acknowledgements

The author wants to thank Vincent T'Kindt and Jean-Charles Billaut for the honor of being invited to write this review and for their patience; the author wants to thank Marjan van den Akker for helpful comments.

References

[1] J.O. Achugbue and F.Y. Chin (1982). Scheduling the openshop to minimize mean flow time. *SIAM Journal on Computing 11*, 665-679.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin (1993). *Network Flows - Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs.

[3] J.M. Van den Akker, J.A. Hoogeveen, and S.L. Van de Velde (2002). Combining column generation and Lagrangean relaxation to solve a single-machine common due date problem. *INFORMS Journal on Computing 14*, 37-51.

[4] B. Alidaee and A. Ahmadian (1993). Two parallel machine sequencing problems involving controllable job processing times. *European Journal of Operational Research 70*, 335-341.

[5] B. Alidaee and I. Dragan (1997). A note on minimizing the weighted sum of tardy and early completion penalties in a single machine: A case of small common due date. *European Journal of Operational Research 96*, 559-563.

[6] J.A. Aslam, A. Rasala, C. Stein, and N. Young (1999). Improved bicriteria existence theorems for scheduling. *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, 846-847.

[7] U. Bagchi (1989). Simultaneous minimization of mean and variation of flow time and waiting time in single machine systems. *Operations Research 17*, 381-386.

[8] U. Bagchi, R.S. Sullivan, and Y.L. Chang (1986). Minimizing mean absolute deviation of completion times about a common due date. *Naval Research Logistics Quarterly 33*, 227-240.

[9] K.R. Baker and G.D. Scudder (1990). Sequencing with earliness and tardiness penalties: a review. *Operations Research 38*, 22-36.

[10] K.R. Baker and J.C. Smith (2003). A multiple-criterion model for machine scheduling. *Journal of Scheduling 6*, 7-16.

[11] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie (2000). Multiprocessor scheduling with rejection. *SIAM Journal on Discrete Mathematics 13*, 64-78.

[12] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali (1990). *Linear Programming and Network Flows*. Wiley, New York.

[13] K. Bülbül, P. Kaminsky, and C. Yano (2001). *Preemption in Single Machine Earliness/Tardiness Scheduling.* Submitted to *Naval Research Logistics* for publication. Department of IEOR, University of California at Berkeley.

[14] K. Bülbül, P. Kaminsky, and C. Yano (2003). *Flow shop scheduling with earliness, tardiness, and intermediate inventory holding costs.* Department of IEOR, University of California at Berkeley.

[15] S. Chakrabarti, C.A. Phillips, A.S. Schulz, D.B. Shmoys, C. Stein, J. Wein (1996). Improved scheduling algorithms for minsum criteria. F. Meyer auf der Heide and B. Monien, (eds.). *Lecture Notes in Computer Science, Springer, Berlin, 1099,* 646-657.

[16] C.-L. Chen and R.L. Bulfin (1990). Scheduling unit processing time jobs on a single machine with multiple criteria. *Computers and Operations Research 17*, 1-7.

[17] C.-L. Chen and R.L. Bulfin (1993). Complexity of single machine, multi-criteria scheduling problems. *European Journal of Operational Research 70*, 115-125.

[18] Z.L. Chen and W.P. Powell (1999). A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem. *European Journal of Operational Research 116*, 220-232.

[19] T.C.E. Cheng, Z.-L. Chen, and C.-L. Li (1996). Single-machine scheduling with trade-off between number of tardy jobs and resource allocation. *Operations Research Letters 19*, 237-242.

[20] T.C.E. Cheng and N. Shakhlevich (1999). Proportionate flow shops with controllable processing times. *Journal of Scheduling 2*, 253-266.

[21] P. Chrétienne and F. Sourd (2003). PERT scheduling with convex cost functions. *Theoretical Computer Science 292*, 145-164.

[22] R.L. Daniels and R.J. Chambers (1990). Multiobjective flow-shop scheduling. *Naval Research Logistics 40*, 85-101.

[23] M.I. Dessouky, B.J. Lageweg, J.K. Lenstra, and S.L. Van de Velde (1990). Scheduling identical jobs on uniform parallel machines. *Statistica Neerlandica 44*, 115-123.

[24] J. Du and J.Y.-T. Leung (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research 15*, 483-495.

[25] B.T. Eck and M.L. Pinedo (1993). On the minimization of the makespan subject to flowtime optimality. *Operations Research 41*, 797-801.

[26] H. Emmons (1975a). One machine sequencing to minimize mean flow time with minimum number of tardy. *Naval Research Logistics Quarterly 22*, 585-592.

[27] H. Emmons (1975b). A note on a scheduling problem with dual criteria. *Naval Research Logistics Quarterly 22*, 615-616.

[28] H. Emmons (1987). Scheduling to a common due date on parallel uniform processors. *Naval Research Logistics 34*, 803-810.

[29] D.W. Engels, D.R. Karger, S.G. Kolliopoulos, S. Sengupta, R.N. Uma, and J. Wein (2003). Techniques for scheduling with rejection. *Journal of Algorithms 49*, 175-191.

[30] G.W. Evans (1984). An overview of techniques for solving multiobjective mathematical programs. *Management Science 30*, 1268-1282.

[31] T.D. Fry, R.D. Armstrong, and H. Lewis (1989). A framework for single machine multiple objective sequencing research. *Omega 17*, 595-607.

[32] T.D. Fry and G. Keong Leong (1987). Single machine scheduling: a comparison of two solution procedures. *Omega 15,* 277-282.

[33] M.R. Garey, D.S. Johnson, and R. Sethi (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research 1*, 117-129.

[34] M.R. Garey, R.E. Tarjan, and G.T. Wilfong (1988). One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research 13*, 330-348.

[35] T. Gonzalez and S. Sahni (1976). Open shop scheduling to minimize finish time. *Journal of the Association for Computing Machinery 25*, 92-101.

[36] V. Gordon, J.-M. Proth, and C. Chu (2002). A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research 139*, 1-25.

[37] R.L. Graham (1966). Bounds for certain multiprocessor anomalies. *Bell System Technical Journal 45*, 1563-1581.

[38] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics 5*, 287-326.

[39] J.N.D. Gupta, A.M.A. Hariri, and C.N. Potts (1999). Single-machine scheduling to minimize maximum tardiness with minimum number of tardy jobs. *Annals of Operations Research 92*, 107-123.

[40] J.N.D. Gupta, V.R. Neppalli, and F. Werner (2001). Minimizing total flow time in a two-machine flowshop problem with minimum makespan. *International Journal of Production Economics 69*, 323-338.

[41] N.G. Hall (1986). Single-and multiple-processor models for minimizing completion time variance. *Naval Research Logistics Quarterly 33*, 49-54.

[42] N.G. Hall, W. Kubiak, and S.P. Sethi (1991). Earliness-tardiness scheduling problems, II: Deviation of completion times about a common due date. *Operations Research 39*, 847-856.

[43] N.G. Hall and M.E. Posner (1991). Earliness-tardiness scheduling problems, I: weighted deviation of completion times about a common due date. *Operations Research 39*, 836-846.

[44] N.G. Hall and C.N. Potts (2003). Rescheduling for new orders. *Operations Research*, to appear.

[45] J.A. Hoogeveen (1992). *Single-Machine Bicriteria Scheduling*. PhD Thesis, CWI, Amsterdam, The Netherlands. Technology.

[46] J.A. Hoogeveen (1996a). Minimizing maximum promptness and maximum lateness on a single machine. *Mathematics of Operations Research 21*, 100-114.

[47] J.A. Hoogeveen (1996b). Single-machine scheduling to minimize a function of two or three maximum cost criteria. *Journal of Algorithms 21*, 415-433.

[48] J.A. Hoogeveen, H. Oosterhout, and S.L. van de Velde (1994). New lower and upper bounds for scheduling around a small common due date. *Operations Research 42*, 102-110.

[49] J.A. Hoogeveen, M. Skutella, and G.J. Woeginger (2003). Preemptive scheduling with rejection. *Mathematical Programming B 94*, 361-374.

[50] J.A. Hoogeveen and S.L. van de Velde (1991). Scheduling around a small common due date. *European Journal of Operational Research 55*, 237-242.

[51] J.A. Hoogeveen and S.L. van de Velde (1995). Minimizing total completion time and maximum cost simultaneously is solvable in polynomial time. *Operations Research Letters 17*, 205-208.

[52] J.A. Hoogeveen and S.L. van de Velde (1996). A branch-and-bound algorithm for single-machine earliness-tardiness scheduling with idle time. *INFORMS Journal on Computing 8*, 402-412.

[53] J.A. Hoogeveen and S.L. van de Velde (1997). Earliness-tardiness scheduling around almost equal due dates. *INFORMS Journal on Computing 9*, 92-99.

[54] J.A. Hoogeveen and S.L. van de Velde (2001). Scheduling with target start times. *European Journal of Operational Research 129*, 87-94.

[55] J.A. Hoogeveen and G.J. Woeginger (2002). Some comments on sequencing with controllable processing times. *Computing 68*, 181-192.

[56] W.A. Horn (1973). Minimizing average flow time with parallel machines. *Operations Research 21*, 846-847.

[57] G. Huang and A. Lim (2003). A new approach to solve a 2-machine bicriteria flowshop scheduling problem. *Proceedings of the 5th International Workshop on Integration of Artificial Intelligence and Operations Research techniques in Constraints Programming for Combinatorial Optimization, CPAIOR'03.*

[58] C.A.J. Hurkens and M.J. Coster (-). On the makespan of a schedule minimizing total completion time for unrelated parallel machines. Unpublished manuscript.

[59] J.R. Jackson (1955). *Scheduling a production line to minimize maximum tardiness*, Research Report 43, Management Sciences Research Project, UCLA.

[60] A. Janiak and M.Y. Kovalyov (1996). Single machine scheduling subject to deadlines and resource dependent processing times. *European Journal of Operational Research 94*, 284-291.

[61] J.J. Kanet (1981). Minimizing the average deviation of job completion times about a common due date. *Naval Research Logistics Quarterly 28*, 643-651.

[62] Karasakal and Köksalan (2000). A simulated annealing approach to bicriteria scheduling problems on a single machine. *Journal of Heuristics 6*, 311-327.

[63] T. Kawaguchi and S. Kyan (1986). Worst case bound of an LRF schedule for the mean weighted flow time problem. *SIAM Journal on Computing 15*, 1119-1129.

[64] Y.-D. Kim and C.A. Yano (1994). Minimizing mean tardiness and earliness in single machine scheduling problems with unequal due dates. *Naval Research Logistics 41*, 913-933.

[65] M. Köksalan, M. Azizoglu, and S. Kondakci (1998). Minimizing flow time and maximum earliness on a single machine. *IIE Transactions 30*, 192-200.

[66] G.J. Kyparisis and C. Koulamas (2000). Open shop scheduling with makespan and total completion time criteria. *Computers & Operations Research 27*, 15-27.

[67] E.L. Lawler (1982). Scheduling a single machine to minimize the number of late jobs, Preprint, Computer Science Division, University of California, Berkeley.

[68] E.L. Lawler and J. Labetoulle (1978). On preemptive scheduling of unrelated parallel machines with linear programming. *Journal of the Association of Computing Machinery 25*, 612-619.

[69] E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan (1981). Minimizing maximum lateness in a two-machine open shop. *Mathematics of Operations Research 6*, 153-158.

[70] J.K. Lenstra (-). Unpublished manuscript.

[71] J.K. Lenstra, E. Tardos, and D.B. Shmoys (1990). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming 46*, 259-271.

[72] J.Y.-T. Leung and G.H. Young (1989). Minimizing schedule length subject to minimum flow time. *SIAM Journal on Computing 18*, 314-326.

[73] C.J. Liao, C.J. Yu, and C.B. Joe (1997). Bicriterion scheduling in the two-machine flowshop. *Journal of the Operational Research Society 48*, 929-935.

[74] T. Masuda and H. Ishii (1994). Two machine open shop scheduling problem with bi-criteria. *Discrete Applied Mathematics 52*, 253-259.

[75] S.T. McCormick and M.L. Pinedo (1995). Scheduling n independent jobs on m uniform machines with both flowtime and makespan objectives: a parametric analysis. *ORSA Journal on Computing 7*, 63-77.

[76] S. Mohri, T. Masuda, and H. Ishii (1999). Bi-criteria scheduling problem on three identical parallel machines. *International Journal of Production Economics 60-61*, 529-536.

[77] A. Nagar, S.S. Heragu, and J. Haddock (1995). A branch-and-bound approach for a two-machine flowshop scheduling problem. *Journal of the Operational Research Society 46*, 721-734.

[78] R.T. Nelson, R.K. Sarin, and R.L. Daniels (1986). Scheduling with multiple performance measures: the one-machine case. *Management Science 32*, 464-479.

[79] E. Nowicki and S. Zdrzałka (1988). A two-machine flow shop scheduling problem with controllable job processing times. *European Journal of Operational Research 34*, 208-220.

[80] E. Nowicki and S. Zdrzałka (1990). A survey of results for sequencing problems with controllable processing times. *Discrete Applied Mathematics 26*, 271-287.

[81] E. Nowicki and S. Zdrzałka (1995). A bicriterion approach to preemptive scheduling of parallel machines with controllable job processing times. *Discrete Applied Mathematics 63*, 237-256.

[82] M.L. Pinedo (2002). *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall.

[83] C. Rajendran (1992). Two-stage flow shop scheduling problem with bicriteria. *Journal of the Operational Research Society 43*, 871-884.

[84] A. Rasala (1999). *Existence theorems for scheduling to meet two objectives.* Technical Report PCS-TR99-347, Dartmouth College.

[85] S. Sahni (1979). Preemptive scheduling with due dates. *Operations Research 27*, 925-934.

[86] S. Sayin and S. Karabati (1999). A bicriteria approach to the two-machine flow shop scheduling problem. *European Journal of Operational Research 113*, 435-449.

[87] D. Shabtay and M. Kaspi (2003). Minimizing the total weighted flow time in a single machine with controllable processing times. *Computers & Operations Research*, to appear.

[88] J.G. Shanthikumar (1983). Scheduling jobs on one machine to minimize the maximum tardiness with minimum number of tardy. *Computers & Operations Research 10*, 253-265.

[89] F.S. Sivrikaya-Serifoglu and G. Ulusoy (1998). A bicriteria two machine permutation flowshop problem. *European Journal of Operational Research 107*, 414-430.

[90] W.E. Smith (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly 1*, 59-66.

[91] F. Sourd (2001). Preemptive scheduling with two minimax criteria. *Annals of Operations Research 107*, 303-319.

[92] F. Sourd (2002). *Scheduling a sequence of tasks with general completion costs.* Research Report 2002/013, LIP6, France.

[93] F. Sourd (2003a). The continuous assignment problem and its application to preemptive and non-preemptive scheduling with irregular cost functions. *INFORMS Journal on Computing*, to appear.

[94] F. Sourd (2003b). Punctuality and idleness in just-in-time scheduling. Submitted to *European Journal of Operational Research.*

[95] F. Sourd and Kedad-Sidhoum (2003). The one machine problem with earliness and tardiness penalties. *Journal of Scheduling 6*, 533-549.

[96] C. Stein and J. Wein (1997). On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Operations Research Letters 21*, 115-122.

[97] R. Tadei, A. Grosso, and F. Della Croce (2002). Finding the Pareto-optima for the total and maximum tardiness single machine problem. *Discrete Applied Mathematics 124*, 117-126.

[98] V. T'kindt and J.-C. Billaut (2002). *Multicriteria Scheduling: Theory, Models and Algorithms.* Springer, Berlin.

[99] V. T'kindt, J.-C. Billaut, and C. Proust (2001). An interactive algorithm to solve bicriteria scheduling problems on unrelated parallel machines. *European Journal of Operational Research 135*, 42-49.

[100] V. T'kindt, J.N.D. Gupta, and J.-C. Billaut (2003). Two-machine flowshop with a secondary criterion. *Computers & Operations Research 30*, 505-526.

[101] V. T'kindt, N. Monmarché, F. Tercinet, and D. Laugt (2002). An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. *European Journal of Operational Research 142*, 250-257.

[102] E. Torng and P. Uthaisombut (1999). Lower bounds for sprt-subsequence algorithms for nonpreemptive scheduling. *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, 973-974.

[103] A. Tuzikov, M. Makhaniok, and R. Männer (1998). Bicriterion scheduling of identical processing time jobs by uniform processors. *Computers & Operations Research 25*, 31-35.

[104] G.L. Vairaktarakis and C.-Y. Lee (1995). The single machine problem to minimize total tardiness subject to minimum number of tardy jobs. *IIE Transactions 27*, 250-256.

[105] S. Verma and M. Dessouky (1998). Single-machine scheduling of unit-time jobs with earliness and tardiness penalties. *Mathematics of Operations Research 23*, 930-943.

[106] R.G. Vickson (1980A). Two single machine sequencing problems involving controllable job processing times. *AIIE Transactions 12*, 258-262.

[107] R.G. Vickson (1980B). Choosing the job sequence and processing times to minimize total processing plus flow cost on a single machine. *Operations Research 28*, 1155-1167.

[108] G. Wan and B. P.-C. Yen (2002). Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties. *European Journal of Operational Research 142*, 271-281.

[109] G. Wan, B. P.-C. Yen, and C.-L. Li (2001). Single machine scheduling to minimize total compression plus weighted flow cost is NP-hard. *Information Processing Letters 79*, 273-280.

[110] L.N. van Wassenhove and K.R. Baker (1982). A bicriterion approach to time/cost tradeoffs in sequencing. *European Journal of Operational Research 11*, 48-54.

[111] L.N. Van Wassenhove and F. Gelders (1980). Solving a bicriterion scheduling problem. *European Journal of Operational Research 4*, 42-48.

[112] M. Wennink (1995). *Algorithmic Support for Automated Planning Boards*, PhD Thesis, Eindhoven University of Technology.

[113] F. Zhang, G. Tang, and Z.-L. Chen (2001). A $\frac{3}{2}$-approximation algorithm for parallel machine scheduling with controllable processing times. *Operations Research Letters 29*, 41-47.