MATHEMATICS OF OPERATIONS RESEARCH
Vol. 26, No. 1, February 2001, pp. 31–49
*Printed in U.S.A.*

# A POLYNOMIAL ALGORITHM FOR MULTIPROCESSOR SCHEDULING WITH TWO JOB LENGTHS

S. THOMAS McCORMICK,
SCOTT R. SMALLWOOD, AND FRITS C. R. SPIEKSMA

The following multiprocessor scheduling problem was motivated by scheduling maintenance periods for aircraft. Each maintenance period is a job, and the maintenance facilities are machines. In this context, there are very few different types of maintenances performed, so it is natural to consider the problem with only a small, fixed number $C$ of different types of jobs. Each job type has a processing time, and each machine is available for the same length of time. A machine can handle at most one job at a time, all jobs are released at time zero, there are no due dates or precedence constraints, and preemption is not allowed. The question is whether it is possible to finish all jobs. We call this problem the *Multiprocessor Scheduling Problem with C job lengths* (MSP$C$).

Scheduling problems such as MSP$C$ where we can partition the jobs into relatively few types such that all jobs of a certain type are identical are often called *high-multiplicity* problems. High-multiplicity problems are interesting because their input is very compact: The input to MSP$C$ consists of only $2C+2$ numbers.

For the case $C = 2$ we present a polynomial-time algorithm. We show that this algorithm guarantees a schedule that uses the minimum possible number of different one-machine schedules, namely three. Further, we extend this algorithm to the case of machine-dependent deadlines (uniform parallel machines), to a multi-parametric case (that contains the case of unrelated parallel machines), and to some related covering problems. Finally, we give some counterexamples showing why our results do not extend to the case $C > 2$.

**1. Introduction.** Consider the following abstraction of a problem arising from scheduling airline maintenance periods (see Smallwood 1988): Each of the $n$ required maintenances is a job, requiring an integral *processing time*, or *length*. There are $m$ maintenance facilities, or machines, each available for an integral time interval, or length, starting at time zero. We will initially assume that all machines are continuously available during the time interval $[0, D]$, i.e., that they all have length $D$; later we will relax this assumption and consider more general variants. All jobs are released at time zero, there are no due dates or precedence constraints, and preemption is not allowed.

Our problem is merely one of feasibility, namely, is there a schedule which fits all $n$ jobs onto the $m$ machines? In other words, we are asking whether the $n$ jobs can be bin-packed into the $m$ bins of length $D$. Of course, an algorithm solving this feasibility problem can be used as a subroutine in a binary search for the minimum possible feasible $D$, i.e., the minimum *makespan* (see, e.g., Lawler et al. 1993 for a background on scheduling and its terminology). Similarly, a feasibility algorithm could be used to compute the minimum possible feasible $m$, thereby finding the minimum number of bins needed to pack the jobs.

This problem is known as the *Multiprocessor Scheduling Problem* (MSP). A straightforward reduction from Partition shows that MSP is NP-Complete when $m = 2$ (see Garey and Johnson 1979 for this and related concepts in NP-Completeness). Alternatively, a simple

Received June 19, 1998; revised July 26, 2000.
*MSC 2000 subject classification.* Primary: 90B35.
*OR/MS subject classification.* Primary: Production/scheduling; secondary: sequencing/deterministic/multiple machine.
*Key words.* Parallel machine scheduling, high-multiplicity scheduling, polynomial algorithms, bin packing, cutting stock, Hilbert basis, integer roundup property, integer decomposition property.

0364-765X/01/2601/0031/$05.00
1526-5471 electronic ISSN, © 2001, INFORMS

reduction from 3-Partition shows that MSP is Strongly NP-Complete. This reduction shows that MSP is Strongly NP-Complete even when the job lengths are such that each machine has exactly three jobs in any feasible schedule.

The number of distinct maintenances for one airline is typically quite small, so it is natural to consider the variant where the jobs can be classified into a small, fixed number $C$ of distinct job lengths. We call this the *Multiprocessor Scheduling Problem With $C$ job lengths* (MSP$C$).

This problem is closely connected to the classic cutting stock problem (see, e.g., Kolen and Spieksma 1999, Magnusson 1995 and Marcotte 1985). In this context, $D$ is the width of a "raw," $m$ is the number of raws, and job types correspond to widths of "finals." Here it also makes sense to consider a small fixed number of job types.

Scheduling problems such as MSP$C$ where the jobs can be partitioned into relatively few types such that all jobs of a certain type are identical are called *high-multiplicity* problems in Hochbaum and Shamir (1991). High-multiplicity problems are interesting because their input is very compact. Hochbaum and Shamir present algorithms for a number of single machine scheduling problems that are polynomial despite this compact input. Other work on high-multiplicity scheduling problems is reported in Granot and Skorin-Kapov (1993), in Granot, Skorin-Kapov and Tamir (1993), and in Clifford and Posner (1995). In the same spirit, Shallcross (1990) gives a polynomial algorithm to solve a one-machine scheduling problem whose input consists of only three numbers. Another problem related to (MSP$C$) is described in Blazewicz et al. (1994), where each job has one of two given processing times, and instead of a machine deadline $D$, a deadline for each job is specified.

Our work also connects with another thread of research concerning integer programming in two variables and/or finding the integer hull of a polyhedron in two dimensions. Note that these two topics are linked, in that many integer hull algorithms use IP as a subroutine, and once we have the vertices of an integer hull it then becomes easy to do optimization. This goes back at least to Kannan's paper (1980) on two-dimensional IP, and the most recent example we have found is by Harvey (1999) on a two-dimensional convex hull. As we shall see (and as Harvey points out), these methods are closely linked to the study of the Euclidean Algorithm for computing the gcd of two numbers. Indeed, in the context of the complexity theory of parallel algorithms, Shallcross et al. (1998) showed that IP in two dimensions and Euclidean gcd are formally equivalent.

For some of these methods it is important to know that the number of vertices of the integer hull is polynomial in the input size. This was first shown for one constraint by Hayes and Larman (1983), and was later sharpened and extended to an arbitrary number of constraints by Hartmann (1989); it was re-discovered (and slightly generalized) in two dimensions by Harvey (1999). These results on two dimensional IP, convex hull, and numbers of vertices will be important for our algorithms. However, note that our problem is not equivalent to any of these problems, since the natural formulation of our problem is an IP with one variable for each one-machine schedule.

We now introduce the notation we will use to describe instances of MSP$C$. Let $l_1, l_2, \ldots, l_C$ be the distinct job lengths, and define $n_k$ as the number of jobs of length $l_k$, so that $n = \sum_{k=1}^{C} n_k$. We refer to a job of length $l_k$ as a *type $k$* job. Then the natural input for MSP$C$ consists solely of the $l_k$, the $n_k$, $m$, and $D$, i.e., only $2C+2$ numbers. We assume w.l.o.g. that each $l_k \le D$, so that $\log D$ measures the input size of the $l_k$ as well as $D$. We are interested in the case where $C$ is small, so we consider $C$ to be fixed (not part of the input). Then, for an algorithm to be polynomial for MSP$C$, it would have to be polynomial in $\log m$, $\log D$, and $\log n$. (Since MSP$C$ is NP-Complete when $C$ is part of the input, such an algorithm would have to be super-polynomial in $C$.) Leung (1982) considers MSP$C$ and gives an $O(n^{2C-2} \log m)$ dynamic programming algorithm. Since Leung's algorithm depends on $n$ instead of $\log n$ it is not polynomial for MSP$C$, but is only pseudo-polynomial. Note that it is not a priori clear that MSP$C$ is even in NP since a certificate of feasibility apparently consists of $m$ possibly different one-machine schedules, and $m$ is not polynomial in the input size.

The main result in this paper is a polynomial-time algorithm for MSP2. Note that the input of MSP2 consists of only six numbers: $l_1, l_2, n_1, n_2, m$, and $D$, so our algorithm needs to be quite careful to remain polynomial. A corollary to this algorithm is that when a feasible schedule exists, there is a feasible schedule which uses at most three different one-machine schedules.

In §2 we prove a lemma partially characterizing feasibility for arbitrary $C$. Section 3 describes our polynomial algorithm for $C = 2$, which is the case of identical machines. We discuss how to extend this algorithm to the case of uniform machines, which we show is equivalent to machine-dependent deadlines, in §4. Section 5 further extends to a multiparametric variant of MSP$C$ which includes the case of unrelated machines. Then §6 briefly shows how these results also apply to related covering (and other) problems. Finally, §7 gives counterexamples showing why our algorithms do not work for the case $C > 2$; §8 briefly discusses a time-indexed formulation of the problem; and §9 contains the conclusion.

## 2. A lemma for arbitrary $C$.

In this section we derive a basic lemma partially characterizing feasibility of MSP$C$. An important object for our analysis is the set $Q$ of feasible *one-machine schedules*:

$$Q \doteq \{(i_1, i_2, \dots, i_C) \in \mathbf{Z}^C \mid i_1, i_2, \dots, i_C \geq 0 \text{ and } l_1 i_1 + l_2 i_2 + \cdots + l_C i_C \leq D\}.$$

Thus $(i_1, i_2, \dots, i_C) \in Q$ means that it is feasible to schedule $i_1$ type 1 jobs, $i_2$ type 2 jobs, $\dots$, and $i_C$ type $C$ jobs on a single machine of length $D$. For the cutting stock problem, a one-machine schedule is called a "cutting pattern." We denote the convex hull of the points in $Q$ by $P$. Note that $P$ can also be described as the integer hull of the relaxed feasible region

$$K \doteq \{(x_1, x_2, \dots, x_C) \in \mathbf{R}^C \mid x_1, x_2, \dots, x_C \geq 0, l_1 x_1 + l_2 x_2 + \cdots + l_C x_C \leq D\},$$

which is called a *fractional knapsack polytope*, with $P$ being the corresponding *knapsack polytope*. Recall that a *simplex* is a full-dimensional polyhedron in $\mathbf{R}^C$ with $C+1$ vertices. We call a simplex with integral vertices in $P$ *unimodular* when it has volume $1/C!$. The key lemma motivating the algorithm in §3 is the following:

LEMMA 2.1.

(a) *If there exist integer points $i^k \in P, k = 1, 2, \dots, C+1$, such that the simplex $T$ spanned by these points is unimodular, and such that the point $M \doteq (n_1/m, n_2/m, \dots, n_C/m) \in T$, then the instance of MSPC is feasible.*

(b) *As a partial converse, if the point $M$ is not in $P$, then the instance of MSPC is infeasible.*

PROOF. (a) Define $B$ to be the $(C+1) \times (C+1)$ matrix whose columns are the $C+1$ vertices of $T$ with an extra 1 added in row $C+1$, i.e.,

$$B \doteq \begin{pmatrix} i^1 & i^2 & \cdots & i^{C+1} \\ 1 & 1 & \cdots & 1 \end{pmatrix}.$$

It is well known that the volume of the simplex $T$ spanned by the points $i^k$, $k = 1, 2, \dots, C+1$, equals $(1/C!)\,|\det(B)|$. Thus $T$ being unimodular is equivalent to $\det(B) = \pm 1$, i.e., $B$ is a unimodular matrix.

Now $M \in T$ implies that there are multipliers $\lambda_k \geq 0$ with $B\lambda = \binom{M}{1}$. With $x_k = m \cdot \lambda_k$, this is equivalent to

$$Bx = \begin{pmatrix} n_1 \\ n_2 \\ \vdots \\ n_C \\ m \end{pmatrix}.$$

Since $B$ is unimodular, and the $n_k$ and $m$ are integral, Cramer's rule ensures that the $x_k$ are integral. Further, $\lambda_k \geq 0$ implies $x_k \geq 0$. Since $x_k$ is integral and nonnegative, we can think of it as the number of times we use one-machine schedule $k$ in an overall schedule. Thus the $x_k$ constitute a feasible solution to the instance of MSP$C$.

(b) Suppose the instance is feasible, and let $b_k^i$ denote the number of type $k$ jobs on machine $i$. Then we have

$$b_k^1 + b_k^2 + \cdots + b_k^m = n_k, \qquad k = 1, 2, \ldots, C.$$

Summing and dividing by $m$ shows that $(n_1/m, n_2/m, \ldots, n_C/m) = M$ can be written as a convex combination of the points $(b_1^i, b_2^i, \ldots, b_C^i)$, $i = 1, \ldots, m$. These points are in $P$, and so $M$ is in $P$. $\square$

It is easy to construct instances of MSP$C$ requiring $C+1$ different one-machine schedules. Since the solution constructed in the proof of Lemma 2.1 uses at most $C+1$ different one-machine schedules, it is in this sense minimal.

Lemma 2.1 says that if we can find $C+1$ integral points in $P$ which induce a unimodular simplex containing $M$, then our instance is feasible, whereas if $M \notin P$, the instance of MPS$C$ is infeasible. So, a natural strategy to decide feasibility is: First decide if $M \in P$, and if yes, look for a unimodular simplex containing $M$. If we find a unimodular simplex $T$, the proof of Lemma 2.1(a) shows how to use $T$ to compute a feasible solution to our instance. Section 3 presents such an algorithm for the case $C = 2$. Moreover, we will show that this algorithm has time-complexity $O((\log D)^2)$, which is polynomial in the input size. A corollary of the algorithm will be that the converses of Lemma 2.1(a) and 2.1(b) are true (in case $C = 2$) because the algorithm will be able to find a unimodular simplex (triangle) containing any point in $P$.

## 3. A polynomial algorithm for two job lengths.

In this section we concentrate on the case of two distinct job lengths, with each machine being available in the interval $[0, D]$, i.e., MSP2. An instance of MSP2 is specified by six numbers: $l_1, l_2, n_1, n_2, m$ and $D$. Lemma 2.1 implies that we can solve MSP2 if we can develop an algorithm which finds three integral points in $P$ inducing a unimodular triangle containing the point $M \doteq (n_1/m, n_2/m)$. This section develops such an algorithm.

In order to determine whether $M \in P$, we will need to know that $P$ has only a logarithmic number of vertices, and we will need to be able to compute all these vertices in polynomial time. We quote here a result of Hartmann (1989) fulfilling these needs (see also Cook et al. 1992). We give a more general version of Hartmann's result than we will need in this section; the more general version will be needed in §5. Let $R$ be the integer hull of the set

$$\big\{(x_1, x_2, \ldots, x_C) \in \mathbf{R}^C \mid x_1, x_2, \ldots, x_C \geq 0;$$
$$l_{1t}x_1 + l_{2t}x_2 + \cdots + l_{Ct}x_C \leq D_t, t = 1, 2, \ldots, s\big\}.$$

Define $D_{\max} = \max_t D_t$. We assume that $0 \leq l_{jt} \leq D_{\max}$ for all $j$ and $t$, so that the input size of each inequality determining $R$ is $O(C \log D_{\max})$.

THEOREM 3.1. *The number of vertices of $R$ is at most $2s^C(6C^3 \log D_{\max})^{C-1}$ (Cook et al.* 1992, *Theorem* 2.1). *Thus, when $s$ and $C$ are fixed, $R$ has only $O((\log D_{\max})^{C-1})$ vertices. These vertices can all be computed in $O((s \log D_{\max})^{2C})$ time using Lenstra's Algorithm* (1983) *for integer programming with a fixed number of variables* (*Hartmann* 1989). □

When we specialize this to the case $C = 2$ (the case of interest in this section), but we keep $s$ for use in §5, we find that the vertices of $P$ can be computed in $O((s \log D)^4)$ time. If we replace Lenstra's Algorithm with Kannan's Algorithm (1980) for integer programming in two variables, the bound speeds up to $O((s \log D)^2)$. Even better, we can and will use Harvey's Algorithm (1999) to directly compute these vertices in only $O(s \log D)$ time. Theorem 3.1 says that the number of vertices of $P$ is $O(s^C(\log D)^{C-1})$. Bárány, Howe, and Lovász (1992) generalize examples dating back to Rubin (1970) (see Schrijver 1986, Remark 18.1) to show that this bound is optimal for fixed $s$ (i.e., there exist instances with $O((\log D)^{C-1})$ vertices). This specializes to $O(s^2 \log D)$ when $C = 2$, but Harvey improves this to $O(s \log D)$, and shows that this is optimal.

We now informally sketch how our algorithm works. First, we use Harvey's Algorithm to compute all vertices of $P$, so that we can check whether $M$ is in $P$. If not, the instance is infeasible by Lemma 2.1(b). Otherwise we invoke a procedure which either finds a unimodular triangle containing $M$, or identifies a smaller polytope $P' \subset P$ containing $M$ for which the procedure is repeated. Polytope $P'$ will be at most half as large as $P$ in at least one dimension, so we need at most a logarithmic number of iterations before finding a unimodular triangle containing $M$. Each iteration of the algorithm takes only logarithmic time, so we get an overall polynomial bound. The algorithm will need the operation of taking the floor of a rational number. In every case the rational will be bounded above by $D$, so this can be done with binary search in $O(\log D)$ time. To facilitate the description of the algorithm let us define an operation $\wedge$ as follows: Given two points $u$ and $v$ in $\mathbf{R}^2$ with $u = (u_1, u_2)$ and $v = (v_1, v_2)$ let $u \wedge v = (\min(u_1, v_1), \min(u_2, v_2))$.

An important concept in the description of this algorithm is an *axial triangle*. This is a right triangle with integral vertices whose right-angle sides are aligned with the $x$ and $y$ axes. Most of our axial triangles will have vertices $p = (p_1, p_2)$ and $q = (q_1, q_2)$ on the hypotenuse with $p_1 < q_1$ and $p_2 > q_2$, and third vertex $p \wedge q$ at the right angle.

Here is the description of the algorithm, called TRIANGLE; justification of some of the claims made in the description are in the proof below.

**Algorithm** TRIANGLE

*Step* 1. Compute the $O(\log D)$ vertices of $P$ using Harvey's Algorithm, which costs $O(\log D)$ time. Let us refer to these vertices as $p_0 \doteq (0, 0), p_1, p_2, \ldots, p_r$, so $P = \operatorname{conv}\{p_0, p_1, p_2, \ldots, p_r\}$.

*Step* 2. (Check whether $M$ is in $P$.) Sort the nonzero vertices of $P$ by a nondecreasing $x$-coordinate and if a tie occurs by decreasing the $y$-coordinate (cost: $O(\log D \log\log D)$; and assume w.l.o.g. that $p_1, p_2, \ldots, p_r$ is the sequence found). Compute, for each $i = 2, \ldots, r$ the line determined by the points $p_{i-1}$ and $p_i$ (cost: $O(1)$/pair), and so get the $O(\log D)$ lines determining the facets of $P$. We can then check whether the point $M$ is on the correct side of each of these lines (cost: $O(1)$/line), and, therefore, if $M \in P$, at a total cost of $O(\log D \log\log D)$ time. If $M \notin P$ STOP; the instance is infeasible by Lemma 2.1.

*Step* 3. (Check if $M$ is in a trivial unimodular triangle.) If $M = 0$ then it is in the unimodular triangle spanned by $(0, 0), (1, 0)$, and $(0, 1)$. If $n_1 = 0$, $n_2 > 0$, and $\lceil M \rceil + (1, 0) \in P$, then $M$ is in the unimodular triangle spanned by $\lceil M \rceil$, $\lceil M \rceil - (0, 1)$, and $\lceil M \rceil + (1, 0)$; the same is true if $n_2 = 0$, $n_1 > 0$, and $\lceil M \rceil + (0, 1) \in P$. Finally, if $n_1, n_2 > 0$, check if $\lceil M \rceil \in P$. If so, then $M$ is in either the unimodular triangle spanned by $\lceil M \rceil - (1, 1)$, $\lceil M \rceil - (1, 0)$, and $\lceil M \rceil - (0, 1)$, or the unimodular triangle spanned by $\lceil M \rceil$, $\lceil M \rceil - (1, 0)$, and $\lceil M \rceil - (0, 1)$. If we find that $M$ is in one of these trivial unimodular triangles we STOP. This costs $O(\log D \log\log D)$ time as in Step 2.

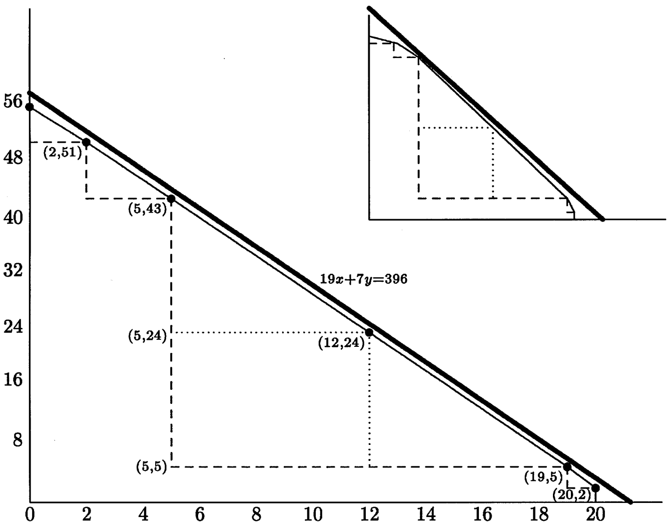

FIGURE 1. The knapsack polytope and its integer hull. The heavy line is the constraint $l_1x+l_2y \le D$. For clarity we have pushed out this line and scaled up the $x$-axis by a factor of four. The light solid line is the integer hull of the knapsack polytope. The dashed lines are the axial triangles formed by its vertices, which are considered in Step 4 of TRIANGLE. Note that the largest dashed triangle has dimensions $14 \times 38$ and $\gcd(14, 38) = 2$, so Step 5 decomposes it into the two $7 \times 19$ dotted triangles, each of which has no integer point on its hypotenuse other than its vertices. Many of the angles and triangles in these figures are too skinny to easily see, so in these figures we show a stylized version of the picture at the upper right to aid understanding. FIGURES 3 and 2 show the next two major steps that TRIANGLE would take when $M$ is in one of the dotted triangles.

*Step* 4. Consider the set of $O(\log D)$ axial triangles spanned by the following triples $(p_{i-1}, p_i, p_{i-1} \wedge p_i)$, $i = 2, \dots, r$ (see Figure 1). By Step 3, $M$ must be in the union of those $O(\log D)$ axial triangles. It costs $O(1)$/triangle to determine which of these triangles contains $M$. The total cost is $O(\log D)$ time.

*Step* 5. (Get an axial triangle with relatively prime sides.) Consider the axial triangle $A$ containing $M$ found in Step 4 or in Step 7. Without loss of generality we can translate $A$ so that its vertices are $(0,0)$, $(a,0)$ and $(0,b)$, with $a, b > 0$ and $a, b$ integer. Compute $\gcd(a,b)$ via the Euclidean Algorithm, which costs $O(\log D)$ (see, e.g., Schrijver 1986 for details on the Euclidean Algorithm). If $\gcd(a,b) = 1$ go to Step 6, else we can decompose $A$ into smaller axial triangles as suggested by Figure 1. Let $a' = a/\gcd(a,b)$, $b' = b/\gcd(a,b)$, and denote the (translated) coordinates of $M$ by $x$ and $y$. Compute $z = \lfloor x/a' \rfloor$. This costs $O(\log D)$ because of the floor. By Step 3, $M$ must belong to the union of the smaller axial triangles, implying that $M$ lies in the axial triangle defined by $(a'z, b-b'z)$, $(a'(z+1), b-b'(z+1))$ and $(a'z, b-b'(z+1))$. When we normalize this smaller axial triangle it will have sides of lengths $a'$ and $b'$, which have gcd 1. The total cost is $O(\log D)$ time.

*Step* 6. (Check if $M$ is in a unimodular triangle coming from a "southeast step" or "northwest step.") If $a = b = 1$ we have found a unimodular triangle containing $M$ and we STOP. Otherwise, notice that when we checked if $\gcd(a,b) = 1$ the Euclidean Algorithm also computed nonnegative integers $p$ and $q$ with $pa - qb = 1$ (Schrijver 1986). It follows that, for any $i > 0$, each of the triangles with vertices $(a,0)$, $((i-1)q, b-(i-1)p)$, and $(iq, b-ip)$ is unimodular (as can be easily verified by computing the determinant induced by these three vertices); these triangles result from taking *southeast steps* from $(0,b)$ in the direction $(q,-p)$; see Figure 2. The vertices of these triangles are in $P$ as long as $b - ip \ge 0$, or $i \le \lfloor b/p \rfloor \doteq k_{\text{SE}}$ (indeed, using the equality $pa - qb = 1$, one easily shows that $i \le \lfloor b/p \rfloor$ implies $i \le a/q$). Similarly, for any $i > 0$ each of the triangles with vertices $(0,b)$, $(a+(i-1)(q-a), (i-1)(b-p))$, and $(a+i(q-a), i(b-p))$ is unimodular, and

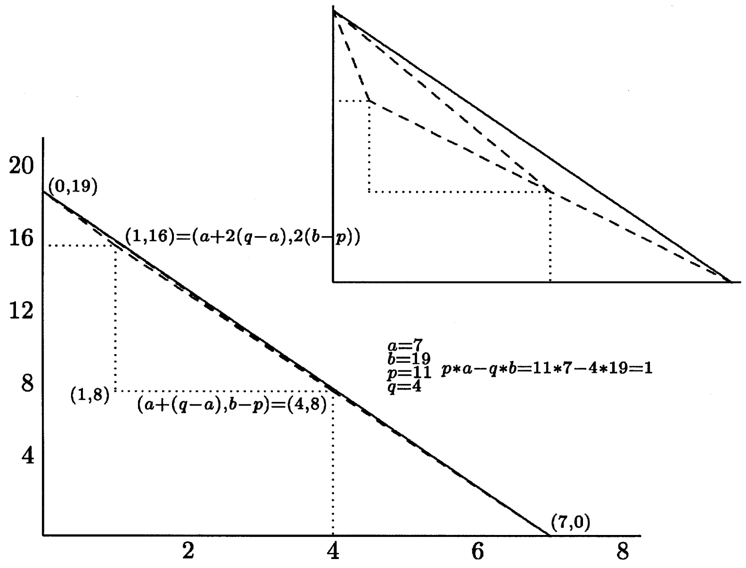

FIGURE 2. Here we apply Step 6 of TRIANGLE to the large $7 \times 19$ axial triangle of Figure 1. In this case we end up taking two northwest steps from vertex $(7,0)$ in the direction $(q-a, b-p)$, giving the two dashed unimodular triangles. These two unimodular triangles induce the three dotted axial triangles shown.

these result from taking *northwest steps* from $(a,0)$ in the direction $(q-a, b-p)$; see Figure 3. The vertices of these triangles are in $P$ as long as $a+i(q-a) \geq 0$, or $i \leq \lfloor a/(a-q) \rfloor \doteq k_{\mathrm{NW}}$. Lemma 3.2 shows that $\min\{k_{\mathrm{SE}}, k_{\mathrm{NW}}\} = 1$ and that $k \doteq \max\{k_{\mathrm{SE}}, k_{\mathrm{NW}}\} \geq 2$. Suppose that $k = k_{\mathrm{SE}}$; the other case is similar. Determine if $M$ is in one of the southeast unimodular triangles by first determining if $M$ is in the big triangle spanned by the points $(0,b)$, $(a,0)$ and $(kq, b-kp)$, which is the union of the southeast unimodular triangles. If not, go to Step 7. Otherwise, since there is, in general, an exponential number of southeast unimodular triangles, we cannot search through them explicitly. We instead find out which

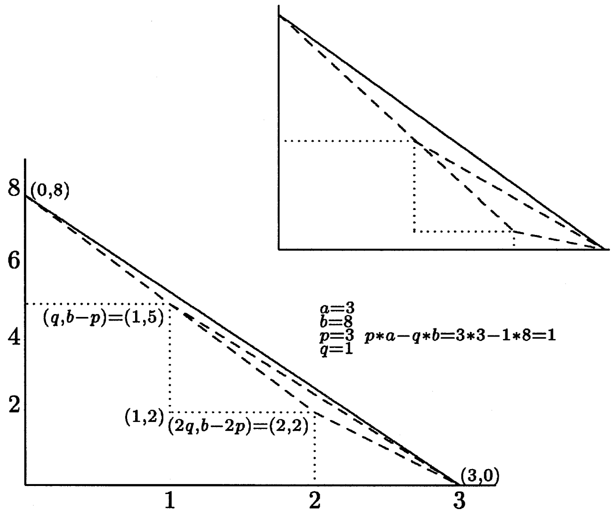

FIGURE 3. Here we apply Step 6 of TRIANGLE to the large $3 \times 8$ axial triangle of Figure 2. In this case we end up taking two southeast steps from vertex $(0,8)$ in the direction $(q, -p)$, giving the two dashed unimodular triangles. These two unimodular triangles induce the three dotted axial triangles shown.

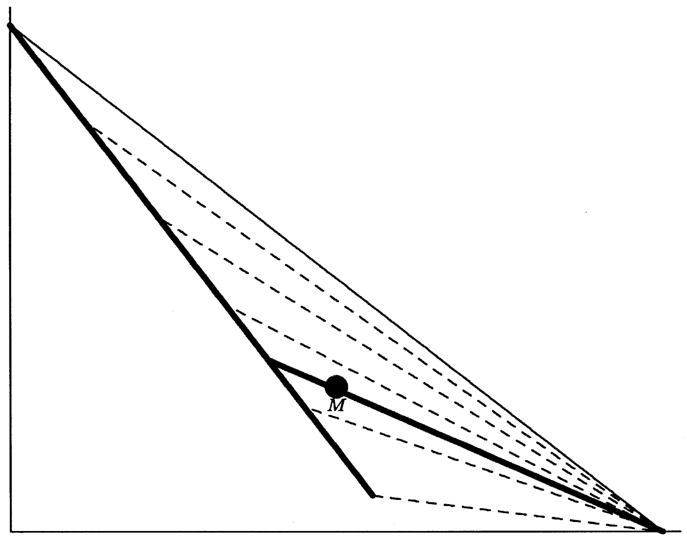

FIGURE 4. In this example of Step 6 of TRIANGLE we have generated five unimodular triangles via southeast steps. We determine which of these unimodular triangles contains $M$ by computing the intersection point of the two heavy lines.

of these triangles contains $M$ indirectly, by computing the $y$-coordinate (denoted by $v$) of the intersection point of the line going through $(0, b)$ and $(q, b-p)$, and the line going through $(a, 0)$ and the (translated) $M$ (see Figure 4). Set $i = \lceil (b-v)/p \rceil$. Then $M$ lies in the unimodular triangle defined by $(a, 0)$, $((i-1)q, b-(i-1)p)$ and $(iq, b-ip)$, so we can STOP. All of this costs $O(1)$ except for $O(1)$ floors, so it costs $O(\log D)$ total time.

*Step* 7. (Find a smaller axial triangle containing $M$.) If $M$ failed to be in the union of the southeast unimodular triangles from Step 6 (the case of northwest unimodular triangles is similar), then (by Step 3) it must be in one of the axial triangles spanned by $(iq, b-ip), ((i-1)q, b-(i-1)p)$ and $(iq, b-ip) \wedge ((i-1)q, b-(i-1)p)$, $1 \leq i \leq k$ or in the axial triangle spanned by $(a, 0), (kq, b-kp)$ and $(kq, 0)$ (see Figure 3). By computing $\lfloor x/q \rfloor$ (recall that $x$ is the $x$-coordinate of $M$ (translated)) we determine which of these axial triangles contains $M$. This costs $O(\log D)$ time. Go to Step 5.

Let us now proceed to prove the correctness of TRIANGLE. A crucial idea in this algorithm is the construction of unimodular triangles by taking southeast or northwest steps (see Step 6). Notice that the first unimodular triangle in both the southeast and northwest series has vertices $(a, 0), (0, b)$, and $(q, b-p)$, and since $0 \leq q \leq a$ and $0 \leq p \leq b$ (Schrijver 1986), this third point is always in $P$, thus showing that we can always feasibly take at least one southeast and northwest step. The next lemma shows that we can take more than one of exactly one of the two steps.

LEMMA 3.2.

(a) *We cannot feasibly take both two southeast steps and two northwest steps.*

(b) *As long as $a+b > 2$, we must be able to feasibly take more than one southeast or northwest step.*

PROOF.

(a) If we can take two or more southeast steps, then $b-2p \geq 0$, which implies that $p \leq b/2$. If we can take two or more northwest steps then $a + 2(q-a) = 2q - a \geq 0$, which implies that $q \geq a/2$. But then $1 = pa - qb \leq ab/2 - ab/2 = 0$, a contradiction.

(b) If we cannot take two southeast steps, then $b-2p<0$, or $b-2p\le -1$, implying that $p\ge (b+1)/2$. Similarly, if we cannot take two northwest steps, then $q\le (a-1)/2$. Then $1=pa-qb\ge ((b+1)/2)a-((a-1)/2)b=\frac{1}{2}(a+b)$, or $a+b\le 2$. □

LEMMA 3.3. TRIANGLE *is correct and runs in* $O((\log D)^2)$ *time.*

PROOF. In Step 5 with axial triangle $A$, notice that $\gcd(a,b)>1$ if and only if there are integer points on the hypotenuse of $A$ other than $A$'s vertices. The new axial triangle found in Step 5 has side lengths $a'$ and $b'$, and these are indeed relatively prime.

Once we get past Step 3, after $O(\log D)$ time we have either found a unimodular triangle containing $M$ (in Step 6), or we have reduced our problem from the original axial triangle $A$ in Step 5 to a similar problem on a smaller axial triangle (the one found in Step 7, with which we loop back to a new iteration of Step 5). In the southeast step case, Lemma 3.2(b) says that at least two smaller axial triangles fit into the $y$-height or $A$, so the $y$-height of each of the smaller axial triangle is at most half of $A$'s $y$-height. Similarly, in the northwest step case, the $x$-width of each smaller axial triangle is at least halved. Since the height and width of our original axial triangle is at most $D$, and since each iteration at least halves one dimension, the algorithm can take at most $O(\log D)$ iterations before termination. Each iteration of Steps 5–7 costs $O(\log D)$ time, so all operations after computing the vertices of $P$ cost only $O((\log D)^2)$ time, which dominates the time for Steps 1–3. Thus over all our algorithm is $O((\log D)^2)$, which is indeed polynomial in the input size. □

A corollary of Lemma 2.1 and Lemma 3.3 is:

COROLLARY 3.4. *MSP2 can be solved in* $O((\log D)^2)$ *time. Moreover, if a solution exists, there is one using at most three different one-machine schedules.* □

Mark Hartmann (1998) has pointed out that there are other ways besides TRIANGLE that one could find a unimodular triangle containing $M$, with the same running time. An advantage of our approach here is that it gives a nearly canonical partition of $P$ into unimodular triangles. Note that if $M$ is very close to the hypotenuse $(a,0)$–$(0,b)$ of an axial triangle, then it can be shown (using essentially the same proof as in Figure 7) that the only unimodular triangle containing $M$ has vertices $(a,0)$, $(0,b)$, and $(q,b-p)$. This shows that computing gcd's is an essential part of any algorithm for solving MSP2, and gives a rough serial algorithm equivalent to the result of Shallcross et al. (1988).

Most instances require using at least three different one-machine schedules, so this is the best possible result. A natural generalization of this to $C>2$ would be that when a solution exists, there always exists a solution using at most $C+1$ different one-machine schedules. Section 7 shows that this conjecture is false even for $C=3$. Thus MSP2, which has a compact input, also has a compact output; we do not know if this weaker result generalizes to $C>2$ or not.

Algorithm TRIANGLE implies:

THEOREM 3.5. *A two-dimensional knapsack polytope $P$ whose vertices are integral can be partitioned into unimodular triangles. Furthermore, given a point $M$ in $P$, a unimodular triangle from this partition containing $M$ can be found in polynomial time.* □

A corollary of Lemma 2.1(a) and Theorem 3.5 shows that when $C=2$, the converses of Lemma 2.1(a) and (b) are true:

COROLLARY 3.6. *An instance of MSP2 is feasible if and only if the point* $M=(n_1/m,n_2/m)$ *is in $P$, which is true if and only if $M$ is contained in a unimodular triangle in $P$.* □

Section 7 shows that Corollary 3.6 is false even for MSP3. The first part of Corollary 3.6 is related to the *integer decomposition property* (IDP). In this context, this says that whenever $M \in P$, then there are $m$ integer points in $P$ whose sum is $(n_1, n_2)$. Here the $m$ points of $P$ are the $m$ one-machine schedules. In fact we have shown a stronger version of IDP, namely that we need to use only (multiples of) three distinct integer points of $P$ to sum to $(n_1, n_2)$. The fact that this $P$ has the (usual) IDP is Theorem 3.1 in Marcotte (1985) (though the proof is incorrect in that reference) and Proposition 7.7 in McDiarmid (1983). By a result of Baum and Trotter (1981), this is, in turn, related to the fact that the associated cutting stock problem has the *integer round-up property*; see Marcotte (1985). Marcotte also gives some other classes of knapsack polytopes for which $M \in P$ is sufficient as well as necessary for feasibility; improved versions of these results appear in Magnusson (1995, Chapter 2).

To get a second interpretation, for $C = 2$ and $l_1$, $l_2$ and $D$ fixed, consider the $3 \times |Q|$ matrix $S$ whose columns are the feasible one-machine schedules of $Q$ with a one in row 3 (like the matrix $B$ defined in the proof of Lemma 2.1). Every feasible point $(n_1, n_2, m)^T$ is in the cone generated by the columns of $S$. Corollary 3.6 shows that every integer point in this cone is in fact an *integer* linear combination of the columns of $S$. This is the defining property of a *Hilbert basis* (Schrijver 1986), so Corollary 3.6 says that the columns of $S$ form a Hilbert basis for its cone.

## 4. Machine-dependent deadlines.

When phrased in scheduling terminology, algorithm TRIANGLE from the previous section deals with the case of two job types and identical parallel machines. This section extends TRIANGLE to the case of two job types and uniform parallel machines, that is the case where each machine has a speed $s_i$, $1 \le i \le m$ (as well as a common deadline $D$) such that processing a job of type 1 (type 2) on machine $i$ takes $l_1/s_i$ $(l_2/s_i)$ time. We first establish that specifying a speed for each machine is equivalent to specifying a (machine-dependent) deadline $D_i$ for each machine (while keeping the original processing times $l_1$ and $l_2$). Indeed, one can easily verify that a schedule that is feasible with respect to machine-dependent deadlines $D_i$ is also a feasible schedule for speeds $s_i = D_i/D_1$, $i = 1, \ldots, m$ and common deadline $D$. Conversely, a schedule that is feasible for speeds $s_i$ and deadline $D$ is feasible for deadlines $s_i D$, $i = 1, \ldots, m$.

Thus we now consider a case with two job types where machine $i$ has length $D_i$, $1 \le i \le m$. We refer to this variant as MSP2$m$. Notice that the input to MSP2$m$ consists of $m + 5$ numbers, so now $m$ can appear in a polynomial bound. We extend the notation of §2 by defining

$$Q_i \doteq \{(i_1, i_2) \in \mathbf{Z}_+^2 \mid l_1 i_1 + l_2 i_2 \le D_i\}, \qquad i = 1, \ldots, m,$$

and the convex hull of points in $Q_i$ by $P_i$. Let $D_{\max} = \max_i D_i$. Harvey's Algorithm can compute the $r_i + 1 = O(\log D_{\max})$ vertices of each $P_i$ in $O(\log D_i)$ time. Thus, we can write:

$$P_i \doteq \operatorname{conv}\{(0, 0), p_1^i, p_2^i, \ldots, p_{r_i}^i\},$$

where $p_j^i$ is a vertex of $P_i$. Note that Harvey's Algorithm orders the nonzero vertices of a knapsack polytope in $\mathbf{R}_+^2$ by nondecreasing $y$-coordinate, and if a tie occurs, by decreasing $x$-coordinate. Then we define *consecutive vertices* to be a consecutive pair in this ordering. Consecutive pairs induce all the "outside" facets of the polytope. Define the *slope* of a consecutive pair $p_j$, $p_{j+1}$ to be the absolute value of the slope of the line containing the two vertices, denoted $m(p_j)$. The ordering of the vertices implies that $m(p_j)$ is monotone decreasing in $j$.

In order to solve MSP2$m$, we first investigate the case of 2 machines, i.e., MSP22. It is clear that any feasible point $(n_1, n_2)$ for MSP22 must belong to the sum of $P_1$ and $P_2$, defined as

$$P^{(2)} \doteq P_1 + P_2 = \{z \in \mathbf{R}_+^2 \mid \exists\, x, y \in \mathbf{R}^2, x \in P_1, y \in P_2 \text{ with } z = x + y\}. \tag{4.1}$$

We will show later (Corollary 4.2) that the condition $(n_1, n_2) \in P^{(2)}$ is also sufficient for feasibility. Thus we will need to know the vertices of $P^{(2)}$ in order to check if $(n_1, n_2)$ belongs to it.

It is well known that $P^{(2)}$ is again a polytope, and that each of its vertices is the sum of a vertex from $P_1$ and a vertex from $P_2$. By our assumption on the ordering of the vertices of $P_1$ and $P_2$, $p_1^1$ $(p_1^2)$ is the right-most vertex of $P_1$ $(P_2)$. Thus it is clear that the right-most vertex of $P^{(2)}$ is $p_1^1 + p_1^2$. Now, if the slope $m(p_1^1)$ is larger than the slope $m(p_1^2)$, then it is clear that the next vertex of $P^{(2)}$ is $p_2^1 + p_1^2$; if instead $m(p_1^1) < m(p_1^2)$, then the next vertex of $P^{(2)}$ is $p_1^1 + p_2^2$; lastly, if $m(p_1^1) = m(p_1^2)$, then the next vertex of $P^{(2)}$ is $p_2^1 + p_2^2$. Continuing in this way gives the following algorithm to construct the vertices $p_j^{(2)}$ of $P^{(2)}$. Variables cnt1 and cnt2 step through the vertices of $P_1$ and $P_2$, and isum indexes the vertices of the sum.

Algorithm CONSTRUCT

**INPUT:** $p_j^1, m(p_j^1), p_j^2, m(p_j^2)$ for all $j$.

```
begin
  $p_1^{(2)} := p_1^1 + p_1^2$;
  cnt1 := 1;
  cnt2 := 1;
  isum := 0;
  while cnt1 $< r_1$ or cnt2 $< r_2$ do
  begin
    isum := isum + 1;
    if $m(p_{\text{cnt1}}^1) > m(p_{\text{cnt2}}^2)$ then
    begin
      cnt1 := cnt1 + 1;
      $p_{\text{isum}}^{(2)} := p_{\text{cnt1}}^1 + p_{\text{cnt2}}^2$;
    end
    else if $m(p_{\text{cnt1}}^1) < m(p_{\text{cnt2}}^2)$ then
    begin
      cnt2 := cnt2 + 1;
      $p_{\text{isum}}^{(2)} := p_{\text{cnt1}}^1 + p_{\text{cnt2}}^2$;
    end
    else begin
      cnt1 := cnt1 + 1;
      cnt2 := cnt2 + 1;
      $p_{\text{isum}}^{(2)} := p_{\text{cnt1}}^1 + p_{\text{cnt2}}^2$;
    end;
  end;
end.
```

REMARK. CONSTRUCT can be extended to construct the vertices of the sum of two arbitrary convex polyhedra in the positive orthant of $\mathbf{R}^2$ having $(0,0)$ as a vertex when we have explicit lists of their vertices. We will need this in §5.

CONSTRUCT only shows us that we can find a description of $P^{(2)}$ in $O(\log D_{\max})$ time. We still must show that any integral point in $P^{(2)}$ corresponds to a feasible schedule. Observe that this does not follow from the definition of $P^{(2)}$: it is conceivable that some integral point $z \in P^{(2)}$ exists which cannot be decomposed into *integral* points $x \in P_1$, $y \in P_2$ with $z = x + y$.

We now give our algorithm for MSP22, which uses CONSTRUCT as a subroutine.

**Algorithm** TRIANGLE2

*Step* 1. Call CONSTRUCT to compute the $O(\log D_{\max})$ vertices $p_j^{(2)}$ of $P^{(2)}$, which costs $O(\log D_{\max})$ time. Determine whether $(n_1, n_2)$ is in $P^{(2)}$. If not STOP; the instance is infeasible.

*Step* 2. Determine if the point $(n_1, n_2)$ is in one of the axial triangles determined by consecutive pairs of the $p_j^{(2)}$. As in Step 4 of TRIANGLE, this can be done in $O(\log D_{\max})$ time. If $(n_1, n_2)$ does not belong to one of these axial triangles, then there is a vertex $p_j^{(2)}$ such that $(n_1, n_2) \le p_j^{(2)}$. From CONSTRUCT we know that $p_j^{(2)}$ is a sum of two vectors, one a feasible schedule from $P_1$, the other a feasible schedule from $P_2$. Removing jobs from these two schedules as needed gives a feasible schedule achieving the multiplicities $n_1$ and $n_2$, so we can STOP.

*Step* 3. Now $(n_1, n_2)$ is in an axial triangle $A$. From CONSTRUCT we know that the two hypotenuse vertices of $A$ have the form either $p_j^1 + p_k^2$, $p_j^1 + p_{k+1}^2$; or $p_j^1 + p_k^2$, $p_{j+1}^1 + p_k^2$; or $p_j^1 + p_k^2$, $p_{j+1}^1 + p_{k+1}^2$. In the first case, if we subtract $p_j^1$ from $A$ we get an axial triangle $A'$ of $P_2$. The translate of $(n_1, n_2)$ is an integral point in $A'$, and so is a feasible schedule for $P_2$. This schedule plus $p_j^1$ realizes a feasible schedule for $(n_1, n_2)$, so STOP. The second case is similar. In the third case either the point $p_j^1 + p_{k+1}^2 \ge (n_1, n_2)$ (the trivial case), or $(n_1, n_2)$ is contained in an axial triangle isomorphic to one in $P_1$ or $P_2$. Since CONSTRUCT can easily keep track of which case applies to each consecutive pair of $P^{(2)}$, this can be done in $O(1)$ time.

This yields the results:

THEOREM 4.1. *MSP22 can be solved in $O(\log D_{\max})$ time.* □

COROLLARY 4.2. *An instance of MSP22 is feasible if and only if $(n_1, n_2) \in P^{(2)}$.* □

In order to generalize Theorem 4.1 to an arbitrary number of machines, we introduce the set $P^{(m)}$, which again must clearly contain the point $(n_1, n_2)$ in order to be feasible:

$$P^{(m)} \doteq \sum_{i=1}^{m} P_i = \left\{ z \in \mathbf{R}_+^2 \mid \exists\ x_i \in P_i \text{ for all } i = 1, \ldots, m \text{ with } \sum_{i=1}^{m} x_i = z \right\}.$$

Define $P^{(1)} = P_1$. Then the vertices of $P^{(m)}$ can be computed by setting $P^{(k)}$ to be the output of CONSTRUCT on inputs $P^{(k-1)}$ and $P_k$ for $k = 2, 3, \ldots, m$. This involves $m$ calls to Harvey's Algorithm to get the vertices of all the $P_k$, and $m-1$ calls to CONSTRUCT, and so takes $O(m \log D_{\max})$ time. It yields $O(m \log D_{\max})$ total vertices for $P^{(m)}$.

By the same reasoning we have used already, each consecutive pair of $P^{(m)}$ will have $m-1$ vertices from $m-1$ of the $P_i$ in common, plus a consecutive pair from the remaining $P_i$. Thus the straightforward generalization of algorithm TRIANGLE2 will solve this problem. This yields:

THEOREM 4.3. *MSP2$m$ can be solved in $O(m \log D_{\max})$ time.* □

COROLLARY 4.4. *An instance of MSP2$m$ is feasible if and only if $(n_1, n_2) \in P^{(m)}$.* □

Notice that in case $D_i = D$ for all $i$ (see §2), $P^{(m)} = \{mx \mid x \in P\}$ or $P^{(m)} = mP$. The algorithm proposed for MSP2$m$ does not boil down to the one presented in §3 in case $D_i = D$. Here, $m$ different one-machine schedules might be constructed, whereas there we were guaranteed to use at most three one-machine schedules. However, the statements deciding feasibility (Corollary 3.6 and Corollary 4.4) are, of course, equivalent.

An obvious common generalization of the model in this section and the model of the previous section is where we have $t$ classes of machines, with $m_k$ machines of class $k$, and every machine in class $k$ has length $D_k$, $k = 1, \ldots, t$. Thus $\sum_k m_k = m$. Here the polytope of the class $k$ machines is $P^{(m_k)}(D_k) = m_k P(D_k)$, so we just need to apply CONSTRUCT successively to $m_1 P(D_1), \ldots, m_t P(D_t)$ and apply the algorithm of this section to get:

THEOREM 4.5. *An instance of MSP*2 *with $m_k$ machines of length $D_k$, $k = 1, \dots, t$, can be solved in $O(t \log D_{\max})$ time.* □

**5. A multi-parametric generalization.** Recall that algorithm TRIANGLE from §3 deals with the case of two job types and identical parallel machines, and the algorithm of the previous section extends TRIANGLE to the case of two job types and uniform parallel machines. This section extends these results to the case of unrelated parallel machines, where we are given numbers $l_{ij}$ that denote the processing time of a job of type $i$ on machine $j$, $i = 1, 2$, $j = 1, \dots, m$ as well as a common deadline $D$. We first establish that specifying these $l_{ij}$ is equivalent to considering the following multi-parametric version of our problem. Now each machine $j$ has $s$ different resources, where the usage of a job of type $i$ of resource $t$ is $l_i^t$ for $i = 1, 2$ and $t = 1, \dots, s$, and the capacity of machine $j$ for resource $t$ is $D_j^t$, $j = 1, \dots, m$, $t = 1, \dots, s$. Then, by setting $s := m$, $l_i^j := l_{ij}$ for $i = 1, 2$, $j = 1, \dots, m$, and

$$D_j^t = \begin{cases} D & \text{if } t = j, \\ \infty & \text{if } t \neq j, \end{cases}$$

for $t = 1, \dots, s$, $j = 1, \dots, m$, we get the case of unrelated parallel machines.

We refer to this multi-parametric generalization as MSP2$m^s$. Notice that for $s = 1$ this problem reduces to MSP2$m$, and that now both $m$ and $s$ can appear in polynomial bounds.

We further extend the notation of §4 by defining for each machine the set of feasible one-machine schedules

$$Q_j^s \doteq \left\{(i_1, i_2) \in \mathbf{Z}_+^2 \mid l_1^t i_1 + l_2^t i_2 \leq D_j^t, t = 1, \dots, s\right\}, \qquad j = 1, \dots, m,$$

and $P_j^s$ as the convex hull of the vertices in $Q_j^s$. Theorem 3.1 shows that Harvey's Algorithm is able to compute the vertices determining the hull of $P_j^s$ in $O(s \log D_{\max})$ time. Each $P_j^s$ is now *not* a knapsack polytope, but rather the intersection of $s$ knapsack polytopes. However, $P_j^s$ does fulfill the requirements mentioned in the remark after CONSTRUCT. Thus the approach described for $s = 1$ is completely valid for general $s$, given the vertices determining $P_j^s$. We leave it to the reader to verify the following theorem.

THEOREM 5.1. *MSP*2$m^s$ *can be solved in $O(ms \log D_{\max})$ time.*

There is a further common generalization of the models of the three previous sections, where we have both $t$ classes of machines and $s$ parameters. Here the polytope of the class $k$ machines for parameter $j$ is $m_k P^{s_j}(D_k)$. We can then apply the algorithms of this and the previous section to get.

THEOREM 5.2. *An instance of MSP*2 *with $s$ parameters and $m_k$ machines of length $D_k$, $k = 1, \dots, t$, can be solved in $O(ts \log D_{\max})$ time.* □

**6. Extension to covering and other problems.** Algorithm TRIANGLE is related to Pick's Theorem (Coxeter 1961, Pick 1899). In our context, this theorem says that any triangle in $\mathbf{R}^2$ with integer vertices and area at least one must contain an integer point different from its vertices. This was used in the original version of this paper (Smallwood 1988). The following generalization of Theorem 3.5 gives a constructive proof of Pick's Theorem. That is, we can input any triangle $T$ with integral vertices to this procedure, and if the volume of $T$ is larger than $\frac{1}{2}$, it will produce an integer point contained in $T$ different from its vertices.

THEOREM 6.1. *Let $R$ be a polyhedron in $\mathbf{R}^2$ defined by $s$ inequalities with largest coefficient $A$, and let $P$ be the integer hull of $R$. Then $P$ can be partitioned into unimodular triangles. Furthermore, given a point $M$ in $P$, a unimodular triangle from this partition containing $M$ can be found in $O(s \log A + (\log A)^2)$ time.*

PROOF. If $s = 1$ then it is easy to find a unimodular transformation that transforms the single constraint into the $x$-axis and $R$ into the half-plane above the $x$-axis. This half-plane is trivially covered by pairs of unimodular triangles, one for each box $(x, y)$, $(x + 1, y + 1)$. Since the transformation is unimodular, it preserves volume, and hence if we find a unimodular triangle $U$ containing the transformed $M$ (which is easy), the inverse transformation of $U$ will be a unimodular triangle containing the original $M$.

So we can assume that $s \geq 2$, so that there is at least one vertex $v$. We use Harvey's Algorithm to compute the $O(s \log A)$ vertices and edges of $P$ in $O(s \log A)$ time. If $P$ is unbounded, we choose $v$ to be a vertex adjacent to unbounded edge $e_1$ and (possibly also unbounded) edge $e_2$. Harvey (1999, Lemma 4.1) shows that in $O(\log A)$ time we can find a unimodular transformation and a translation such that we can assume w.l.o.g. that $v = 0$, that $e_1$ is the $y$-axis, and that $e_2$ belongs to the positive orthant (see Figure 5). Once again, since this transformation is unimodular, we can work with the transformed problem. Now we use the horizontal line passing through the vertex with largest $y$-coordinate to partition $P$ into a bounded piece $P'$ and an unbounded piece as suggested by Figure 5. Then $P \backslash P'$ partitions into an infinite collection of identical axial triangles, and some rectangles. TRIANGLE can be used to partition a representative of each collection of axial triangles into unimodular triangles, and the rectangles trivially partition into unimodular triangles. If $M \in P \backslash P'$, then it is easy to compute whether $M$ belongs to one of the collections of axial triangles, and if so, to which unimodular triangle within the collection $M$ belongs, and the case where $M$ belongs to one of the rectangles is even easier.

Thus we need worry only about bounded polytopes $P'$. Select an arbitrary vertex $v$ of $P'$ and use it to partition $P'$ into $O(s \log A)$ triangles with integral vertices by adding the line segments from $v$ to every other vertex of $P'$ (see Figure 5). It is easy to compute which of these triangles contains $M$, still within the $O(s \log A)$ time for Harvey's Algorithm.

So our problem is reduced to finding a partition of an arbitrary triangle $T$ with integral vertices into unimodular triangles, and finding which of these triangles contains $M$, in $O((\log A)^2)$ time. Again using Lemma 4.1 from Harvey (1999), in $O(\log A)$ time we can

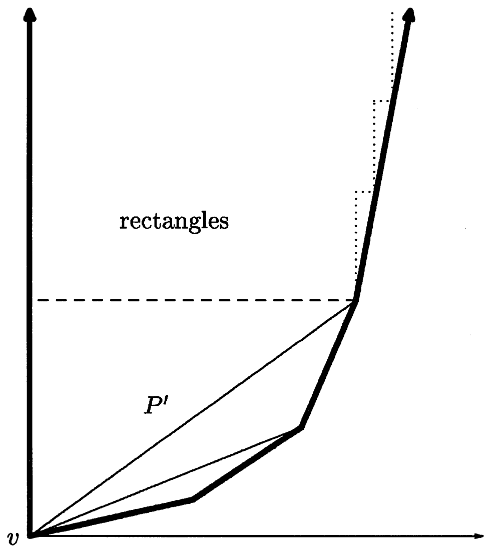

FIGURE 5. When $P$ is unbounded we apply a unimodular transformation that takes $v$ to 0, its adjacent unbounded edge to the $y$-axis, and its other adjacent edge to the positive orthant. The heavy lines show the transformed $P$. We then use the horizontal dashed line to partition $P$ into its bounded part $P'$ and an unbounded part. The unbounded part of $P$ gets partitioned into the dotted collection of axial triangles, and rectangles (not shown). We add the thin solid lines from vertex $v$ in $P'$ to partition its bounded part into (nonaxial) triangles.

find a unimodular transformation and a translation such that we can assume w.l.o.g. that the vertices of $T$ are $(a,0)$, $(0,b)$, and $(0,c)$ with $a,b > 0$ and $c < b$. If $c \leq 0$ then $T$ is composed of two axial triangles and TRIANGLE will solve this in $O((\log A)^2)$ time.

So suppose instead that $0 < c < b$. Compute $k = \gcd(a,b)$ and partition $T$ into $k$ new triangles (see Figure 6), and determine which new triangle contains $M$ as in Figure 4. Then recurse on the new triangle. The volume of this new triangle is at most half the volume of $T$, so we can have at most $O(\log A)$ such iterations throughout the algorithm. Thus we can assume from now on that $\gcd(a,b) = 1$ so that the line $H$ containing $(a,0)$ and $(0,b)$ has no integer points between $(a,0)$ and $(0,b)$.

Now run TRIANGLE starting at Step 6 on the axial triangle $T'$ with vertices $(a,0)$, $(0,b)$, and $(0,0)$. We claim that the vertex $(q, b-p)$ of the first southeast/northwest step must belong to $T'$: Equivalently, if we define $(0,d)$ as the point where the line $L$ containing $(a,0)$ and $(q,b-p)$ hits the $y$-axis ($d$ might not be an integer), then we are claiming that $c \leq d$ (see Figure 7).

Since $(0,b)$, $(a,0)$, and $(q,b-p)$ form a unimodular triangle, all the integer points within $T'$ must lie on the equally spaced series of lines parallel to $H$ whose first member passes through $(q,b-p)$. The intersections of these lines with the part of $L$ within $T'$ are precisely the new vertices for the northeast steps from $(a,0)$. If $c > d$, then the line containing $(0,c)$ which is parallel to $H$ must also contain another integer point $(e,f)$ where it intersects $L$. But now the integer points $(0,c)$, $(e,f)$, and $(a,0)$ are three of the four vertices of a parallelogram whose fourth vertex (see Figure 7) must be an integral point on $H$ between $(0,b)$ and $(a,0)$, a contradiction.

Continue taking southeast or northwest steps as long as we stay in $T'$. If $M$ belongs to one of the generated unimodular triangles, then we can stop. Otherwise we divide the remainder into one skinny triangle and a fan of other triangles (see Figure 8), determine which of these new triangles $M$ belongs to again as in Figure 4, and recurse. By Lemma 3.2 the volume of each of these new triangles is at most half the volume of $T'$, so again we must terminate in at most $O(\log A)$ iterations. □

Here is one application of this theorem. MSP2 is a type of *packing* problem: that is, we are asking if there is some way to pack all the given jobs into the given machines (bins). The lengths of the machines give upper bounds on how much can be packed into one machine. We could instead consider the natural *covering* version of MSP2, where $D$ is now a lower

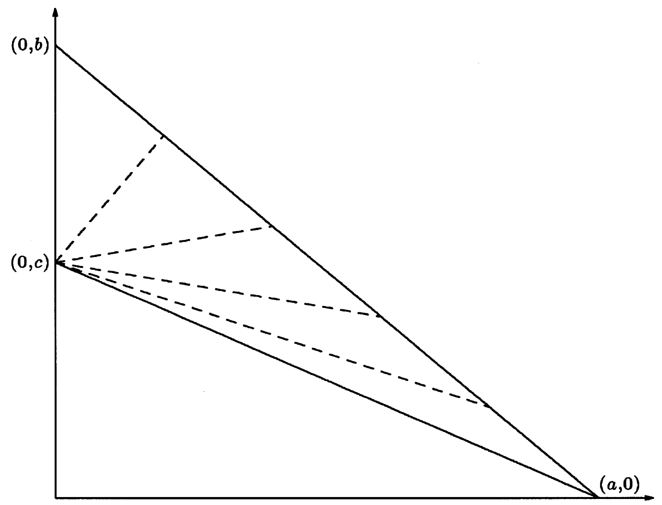

FIGURE 6. When $T$'s hypotenuse contains integer points (here $\gcd(a,b) = 5$, so there are four points on the hypotenuse), we subdivide $T$ into new triangles with common vertex $(0,c)$ (using the dashed lines).

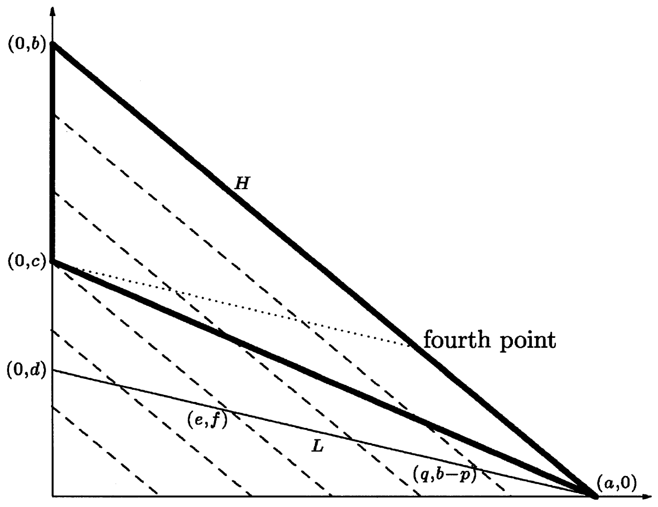

FIGURE 7. Illustration of why point $(q, b-p)$ must belong to $T'$. Here $T'$ is the heavy triangle, and all integer points of $T'$ belong to one of the series of dashed lines parallel to $H$, the first of which passes through $(q, b-p)$. The intersection of each dashed line with $L$ is integral (one of the new northwest step points). Thus $(0, c)$ is on a dashed line, and $(e, f)$ is also integral. Points $(0, c)$, $(e, f)$, and $(a, 0)$ form three integral corners of a parallelogram, so the "fourth point" (determined by the intersection of the dotted line parallel to $L$ with $H$) is also integral, a contradiction.

bound on how much work must be assigned to each machine. That is, given $m$, $D$, $l_1$, $l_2$, $n_1$, and $n_2$, we ask whether there exists an assignment of jobs to machines such that each machine is busy for at least $D$ time units.

This covering version of MSP2 can be solved using Theorem 6.1. In fact, due to the simple structure here, we can solve this variant in $O((\log D)^2)$ time. Furthermore, we can similarly get extensions of the results of §§4 and 5 to the covering case. We can also solve versions where we have both a lower bound $D'$ and and an upper bound $D \geq D'$, and many other such problems.

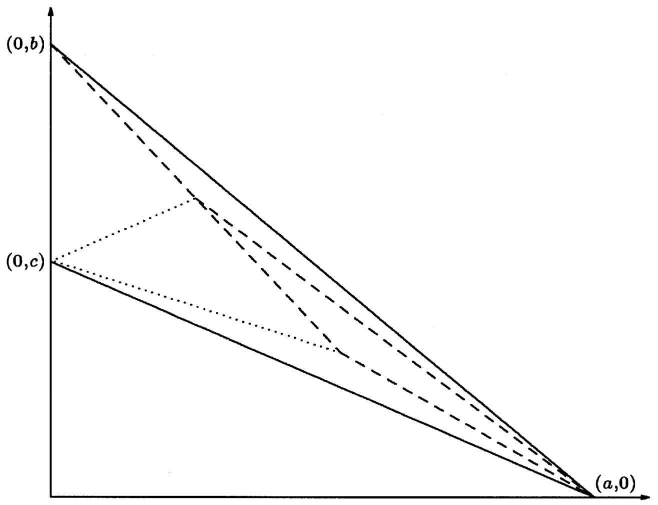

FIGURE 8. Subdividing triangle $T'$ after taking a maximum number of southeast steps. Here we have taken two southeast steps (dashed lines), and the remaining part of $T'$ is then partitioned into three new triangles (dotted lines).

## 7. The case $C>2$: Counterexamples.

We now consider the extent to which our results can be generalized to cases with $C > 2$. We noted that one way to view our algorithm was as a consequence of Pick's Theorem. Unfortunately, in dimensions higher than two, it is easy to find simplices with integer vertices which are not unimodular, but which contain no integer point other than their vertices. In this sense, our algorithm for $C = 2$ fails to extend to higher dimensions because this version of Pick's Theorem is not true in higher dimensions. We now give two examples showing that both parts of Lemma 2.1 do not generalize even to $C = 3$. Both examples will involve points contained in $P$, and thus contained within a simplex with integer vertices in $P$, but which are not contained within any unimodular simplex in $P$. The first example deals with a feasible instance, the second with an infeasible instance.

Lemma 2.1(a) said that if $M$ is contained in a unimodular simplex in $P$ then the instance is feasible, and Corollary 3.6 showed that the converse is true for $C = 2$. Moreover, Corollary 3.4 showed that for a feasible instance of MSP2, there is always a schedule using at most $3 = C + 1$ one-machine schedules. To show that these results do not generalize to higher values of $C$, consider the instance with $C = 3$, $l = (323, 171, 153)$, $D = 2907$, $m = 5$, and $n = (12, 30, 36)$. There is a feasible schedule that uses five different one-machine schedules, namely $(9, 0, 0)$, $(0, 17, 0)$, $(0, 0, 19)$, $(0, 8, 10)$, and $(3, 5, 7)$. The total idle time for these jobs is 21 time units. Exhaustive enumeration of feasible one-machine schedules with idle time at most 21 shows that there is no way to put together any four one-machine schedules to get a total idle time of 21 time units. Thus the point $M$ for this instance cannot belong to a unimodular simplex despite being feasible.

Lemma 2.1(b) said that the point $M$ for a feasible instance must belong to $P$, and Corollary 3.6 showed that the converse is true for $C = 2$. An example of Alan Hoffman shows that Corollary 3.6 is false for MSP3: For $C = 3$, $l = (6, 10, 15)$, $n = (4, 2, 1)$, $m = 2$, and $D = 30$, it is easy to check that the point $M = (2, 1, \frac{1}{2}) \in P$ (since the knapsack polytope $P$ and the fractional knapsack polytope $K$ coincide for this instance). However, it is also easy to see that this instance is none the less infeasible.

## 8. A time-indexed formulation of the problem.

An alternate way to formulate MSP$C$ is with a *time-indexed* formulation. Let $x_{jt}$ be the number of jobs of type $j$ that start at time $t$, $j = 1, \ldots, C$ and $t = 1, \ldots, D - l_j + 1$. The constraints would then be:

$$\sum_{t=1}^{D-l_j+1} x_{jt} = n_j \quad \text{for } j = 1, \ldots, C, \quad \text{and}$$

$$\sum_{j=1}^{C} \sum_{s=t-l_j+1}^{t} x_{js} \le m \quad \text{for } t = 1, \ldots, D;$$

the first constraints say that $n_j$ jobs of type $j$ must be scheduled, and the second say that at most $m$ machines can be in use at one time. Of course, the size of this formulation is exponential in the size of the input data (since it depends linearly on $D$).

Section 3 of Queyranne and Schulz (1994) gives a good survey of time-indexed formulations of scheduling problems. Time-indexed formulations tend to be very large, often with pseudo-polynomial size. On the other hand, they are provably stronger than other formulations for several (but not all) classes of scheduling problems. In the case of MSP$C$, it is possible to prove that a result similar to Corollary 3.6 holds:

THEOREM 8.1 (Smallwood 1988). *There is a fractional solution to the time-indexed formulation of MSP2 if and only if it has an integer solution.* □

In fact, similar results hold also for the generalizations of the problem considered in §§4 and 5; see Smallwood (1988) for further details. Unfortunately, the proofs are nonalgorithmic. The second counterexample in §7 is also a counterexample to Theorem 8.1 for the case $C = 3$.

**9. Conclusions and further work.** We have shown that when we are doing simple parallel machine scheduling with identical machines and two job lengths (or, equivalently, bin packing with two item sizes), a polynomial algorithm exists, despite the fact that the input can be represented very compactly. This special-purpose algorithm is based on a geometric view of the problem that allows us to extend our result to several more complex situations with two job lengths: Parallel machine scheduling with uniform parallel machines (which corresponds to different-sized bins), a multi-parametric variant (which corresponds to the so-called vector-packing problem) that contains parallel machine scheduling with unrelated machines as a special case, and covering variants of the problem are also solvable in polynomial time. Cases where $C > 2$ remain open.

In light of the counterexamples in §7, it is tempting to conjecture that MSP3 is NP-Hard. However, we do not know any counterexamples which require noncompact output (such as Clifford and Posner (1995) have for some parallel $C_{\max}$ problems). Indeed, there is some empirical evidence from some real cutting stock problems in Kolen and Spieksma (1999) that real data does tend to produce quite compact output.

Our interest in this problem was motivated by a real problem in aircraft maintenance. Our algorithm relies on pre-computing the vertices of a knapsack polytope using Harvey's Algorithm (1999). Harvey's algorithm was developed to support applied work in Constraint Logic Programming, and relies on clever application of intermediate results from the Euclidean Algorithm for gcd, so it should be practical. The rest of our algorithm should execute quite fast, so the algorithm should be useful in practice.

In the real world, despite the technical fact that the input can be very compact, any concrete schedule will be carried out using time $D$ and involving $n$ jobs. Thus an algorithm that is polynomial in $n$ and $D$ might then be useful. Such an algorithm is Leung's (1982) (pseudo-polynomial) dynamic program, which runs in $O(n^{2C-2} \log m)$ time. This algorithm has the advantage that it can deal with cases where $C > 2$. By contrast, any generalization of our methods to cases with $C > 2$ would seem to involve having to compute the vertices of the integer hull. This can be done in polynomial time as noted by Theorem 3.1, but it involves using Lenstra's Algorithm (1983) for IP in fixed dimension, which is widely viewed as being impractical.

**Acknowledgments.** Many people have helped us with this paper. The second author thanks Alan Manne (his thesis advisor), and Alan Hoffman for all of their help with Smallwood (1988), which led to this paper. The authors thank: Jan Karel Lenstra for showing us the reference by Leung (1982); Maurice Queyranne and Andreas Schulz for showing us references by Shallcross (1990) and Queyranne and Schulz (1994) and for helpful conversations about §8; Bill Cook for pointing out the results in Cook et al. (1992) and Hartmann (1989); Mark Hartmann and Odile Marcotte for pointing out the relevance of Marcotte (1985) and many helpful discussions; Bill Pulleyblank and David Shallcross for tracking down and sending us the reference by Pick (1899); Les Trotter for showing us references by Baum et al. (1981), Marcotte (1985) and Magnusson (1995), and for helpful conversations; Uday Rao for many stimulating conversations about this problem; and Warwick Harvey for pointing out an error in the original version of Theorem 6.1 and other helpful suggestions. Finally, we thank the reviewers for their comments.

Research of the first author was partially supported by an NSERC Research Grant. The second author was at Stanford OR Department while completing the research. The third author completed some of this research while at the University of British Columbia.

## References

Bárány, I., R. Howe, L. Lovász. 1992. On integer points in polyhedra: A lower bound. *Combinatorica* **12** 135–142.

Baum, S., L. E. Trotter, Jr. 1981. Integer rounding for polymatroid and branching optimization problems. *SIAM J. Algebraic Discrete Methods* **2** 416–425.

Blazewicz, J., R. E. Burkard, G. Finke, G. J. Woeginger. 1994. Vehicle scheduling in two-cycle flexible manufacturing systems. *Math. Comput. Modelling* **20**(2) 19–31.

Clifford, J. J., M. E. Posner. 1995. *Parallel machine scheduling with high multiplicity*. Working paper 1995-006, The Ohio State University, Cincinnati, OH.

Cook, W., M. Hartmann, R. Kannan, C. McDiarmid. 1992. On integer points in polyhedra. *Combinatorica* **12** 27–37.

Coxeter, H. S. M., 1961. *Introduction to Geometry*. John Wiley and Sons,

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.

Granot, F., J. Skorin-Kapov. 1993. On polynomial solvability of the high multiplicity total weighted tardiness problem. *Discrete Appl. Math.* **41** 139–146.

———, ———, A. Tamir. 1993. Using quadratic programming to solve high multuplicity scheduling problems on parallel machines. *Algorithmica* **17** 100–110.

Hartmann, M. 1989. *Cutting planes and the complexity of the integer hull*. SORIE Ph.D. thesis, Cornell University, Ithaca, NY.

———. 1998. Personal communication.

Harvey, W. 1999. Computing two-dimensional integer hulls. *SIAM J. Comput.* **28** 2285–2299.

Hayes, A. C., D. G. Larman. 1983. The vertices of the knapsack polytope. *Discrete Appl. Math.* **6** 135–138.

Hochbaum, D. S., R. Shamir. 1991. Strongly polynomial algorithms for the high multiplicity scheduling problem. *Oper. Res.* **39** 648–653.

Kannan, R. 1980. A polynomial algorithm for the two-variable integer programming problem. *JACM* **27** 118–122.

Kolen, A. W. J., F. C. R. Spieksma. 1999. Solving a bi-criteria cutting stock problem: A case study. Manuscript, Maastricht University, Maastricht, The Netherlands.

Lawler, E. L., J. K. Lenstra, A. H. G. Rinooy Kan, D. B. Shmoys. 1993. Sequencing and scheduling: Algorithms and complexity. S. C. Graves, A. H. G. Rinooy Kan, P. Zipkin, eds. *Handbooks in Operations Research and Management Science*, Volume 4, Chapter 9. North Holland.

Lenstra, H. W., Jr. 1983. Integer programming in a fixed number of variables. *Math. Oper. Res.* **8** 538–548.

Leung, J. Y-T. 1982. On scheduling independent tasks with restricted execution times. *Oper. Res.* **30** 163–171.

Magnusson, S. 1995. *Cutting stock problems: Theory and practice*. Ph.D. thesis, SORIE, Cornell University, Ithaca, NY.

Marcotte, O. 1985. The cutting stock problem and integer rounding. *Math. Programming* **33** 82–92.

McCormick, S. T., S. R. Smallwood, F. C. R. Spieksma. 1996. Polynomial algorithms for multiprocessor scheduling with a small number of job lengths (extended abstract). *Proc. Eighth SODA* 1997 509–517.

McDiarmid, C. 1983. Integral decomposition in polyhedra. *Math. Programming* **25** 183–198.

Pick, G. 1899. Geometrisches zur Zahlenlehre. *Lotos Naturwissenschaftlich Ze.* **47** 315–323 (in German).

Queyranne, M., A. Schulz. 1994. *Polyhedral approaches to machine scheduling*. Report No. 408/1994, Fachbereich Mathematik, Technische Universität, Berlin, Germany.

Rubin, D. S. 1970. On the unlimited number of faces in integer hulls of linear programs with a single constraint. *Oper. Res.* **18** 940–946.

Schrijver, A. 1986. *Theory of Linear and Integer Programming*. John Wiley & Sons, New York.

Shallcross, D. F. 1990. A polynomial algorithm for a one machine batching problem. *Oper. Res. Lett.* **11** 213–218.

———, V. Y. Pan, Y. Lin-Cruz. 1998. Planar integer linear programming is NC equivalent to Euclidean GCD. *SIAM J. Comput.* **27** 960–971.

Smallwood, S. R. 1988. *A linear programming relaxation for scheduling problems.* Stanford Operations Research Ph.D. thesis, Stanford, CA.

S. T. McCormick: Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, British Columbia V6T 1Z2 Canada; e-mail: stmv@adk.commerce.ubc.ca

S. R. Smallwood: Morgan Stanley Dean Witter, 4614 N. Dittmar Rd., Arlington, Virginia 22207 USA; e-mail: scott.smallwood@msdw.com

F. C. R. Spieksma: University of Maastricht, P.O. Box 616, NL-6200 MD, Maastricht, The Netherlands; e-mail: spieksma@math.unimaas.nl