# A Polynomial Algorithm for Lotsizing and Scheduling of Two Products on Parallel Machines

João Eduardo C. Scheidt
Unisoma

Clovis Perin
Unicamp

**Abstract**

A strongly polynomial algorithm for lotsizing and scheduling $N$ orders of two products on $M$ identical parallel machines is proposed; preemption and job splitting are allowed and setup time between tasks of different products is required. Any order implies an arbitrary deadline, and the quantity of product (or products) demanded. Initial machine ready times are arbitrary but machines are assumed to be available continuously thereafter. The aim is to minimize any arbitrary non-decreasing function of the number of setups for each product using a scheduling which satisfies both the demands and deadlines in a make-to-order environment. Wavy schedulings are defined using unbalanced scheduling and scheduling gaps, as these concepts can be useful in the design of heuristics for model extensions.

*Keywords: scheduling theory, complexity, parallel machines, polynomial algorithm*

## 1 Introduction

Scheduling problems involving setups on parallel machines show a lack of theoretical development, although they are a crucial aspect of two new industrial trends. The first is the importance of product customizing and quality improvement in a competitive market. This has led to a reduction in lot sizes in manufacturing systems, which has made setup time and cost

considerations of great economic relevance. However, relatively little work has been done on scheduling problems with machine setups, with even less work involving setup time considerations [17]. As a consequence, poor performance heuristics that ignore setup effects are commonly used [11].

The second trend is an attempt to improve efficiency and productivity through the concept of parallelism. The use of parallel machines not only contributes to production flexibility, but also helps cope with bottleneck problems and improves production regularity [9]. Despite the advantages of the use of parallel machines, however, little work has been done with parallel machines using due-date based performance measures.

One reason for the lack of development may be due to the mathematical intractability of problems of this type, as the use of parallel machines and the choice involved in the assignment of jobs to machines greatly increases the complexity of a problem. Moreover, scheduling problems that involve either setup costs or setup times are notoriously difficult to solve. The consideration of setup times also introduces non-conservation of production capacity into the model, which can make it hard even to judge the feasibility of specific problem instances.

Problems of this kind appear in many industries involving continuous processing. In the textile industry, for example, scheduling is characterized by a large number of identical machines (looms) which require extensive and time-consuming setups. In the paper industry, the number of machines is small and setup time is quite short, but the number of products tends to be large due to the numerous combinations of paper thickness and composition. The flat glass industry is characterized by numerous glass thickness setups and also costly composition setups.

Restriction to jobs of two products on a single machine is present in the works of Lee and Danusaputro [10] and Gupta [7]. Cattrysse [2] works on a single machine DLSP (Discrete Lotsizing and Scheduling Problem) with setup times. Unal and Kiran [18] present an exponential time exact algorithm (polynomial time if the number of setups is bounded); they consider a single machine and an arbitrary number of products. Monma and Potts [12] developed a dynamic programming algorithm for minimizing $Lmax$ (maximum lateness) on parallel machines; it is a polynomial time algorithm for a fixed number of products. A closely related work is the one of Gonçalves and Marques [5] which presents a branch and bound heuristic to solve the problem involving several products and holding costs. Guinet [6] and Serafini and

Speranza [16] both produce approaches for the textile industry environment, although the former does not allow preemption, whereas the latter consider job splitting, but using a special kind of setup.

# 2 Structure of the Algorithm

This paper presents a strongly polynomial algorithm for lotsizing and scheduling $N$ orders of two products on $M$ identical parallel machines so that total setup time is minimal and deadlines are met. Arbitrary non-negative setup times are considered. The number of setups for both products is simultaneously minimized, while neither planning horizons nor restrictions as to the number of machines need to be considered.

### Problem Statement

The proposed algorithm solves scheduling problems of the following type:
*Given*

- $M$ parallel identical machines that work with tasks of two products
- A list of $N$ orders with deadlines and job demands for the products
- The setup length of each product

*Find (if possible)*

- A lotsizing and scheduling of tasks on the machines requiring the least number of setups

*Where*

- Setup is required between tasks of different products on the same machine
- Job demands should be satisfied without delay
- Job splitting is allowed
- Preemption is allowed

A problem is feasible if it has a feasible scheduling. Feasible scheduling allocates tasks and accounts for appropriate setups so that the total quantity of a product which can be produced by a deadline is greater than or equal to the quantity required. Any feasible problem has an optimal scheduling, which is that feasible scheduling which utilizes the least number of setups.

### Scheduling Gap

Prior to the presentation of the algorithm itself, it is necessary to introduce the concept of scheduling gap. For a given machine, this is the time interval $[s, t]$ in which its use is not scheduled, although its use is scheduled for one product before and for the other after this period of time (Figure 1). The time must be more than sufficient for the setup of the second product, so that time can be made available for additional production of either of the two products involved.

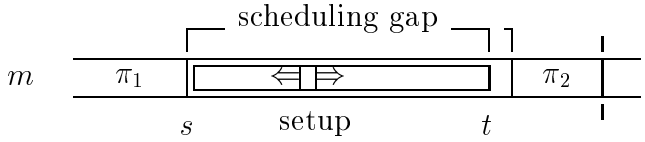


Figure 1: *Example of a Scheduling Gap*

### The Algorithm

*The algorithm iterates on a non-decreasing sequence of deadlines. At each iteration, tasks are allocated to the machines so that the demand for each product will be satisfied by the deadline. Machines are grouped (group 1 or group 2) in accordance with the product ($\pi_1$ or $\pi_2$) being processed. The available processing time of each machine is the difference between the current deadline and its current ready time; moreover, there may be a scheduling gap on one of the machines. The allocation of jobs for the production of product $\pi_1$ proceeds in the following order until its demand is satisfied:*

**a)** *it progressively involves the group 1 machines; the group 1 machine with the smallest non-null available time is selected; allocation is made forwards, from the machine's ready time to the deadline;*

**b)** *further production utilizes the time interval $[s, t]$ of the scheduling gap; depending on the setup required in the scheduling gap, for the same or for the opposite product, allocation is made backwards from $t$ or forwards from $s$;*

**c)** *it progressively involves the group 2 machines; the group 2 machine with the greatest available time is selected; allocation is made backwards from*

*the deadline to the machine's ready time and is preceded by a setup for the product.*

*Each task fully occupies the available time on the machine with the possible exception of the last allocation; if it is made on a group 2 machine, the old scheduling gap is already fully occupied and possibly a new scheduling gap is created on this machine. Tasks involving product $\pi_2$ are allocated in a similar way, except that allocation order starts with group 2, follows with the scheduling gap and finishes with group 1. If the algorithm cannot allocate enough work to satisfy the current demand of any of the products, then it stops because no solution is possible, and, therefore, the problem is infeasible.*

It should be noted that, for each iteration, all new setups are for the same product and are created in a non-decreasing time sequence. The final scheduling utilizes the least number of setups and tends to be unbalanced in terms of new ready times. Moreover, the setup time of a scheduling gap becomes fixed before and earlier than any additional setup time allocation.

**Example**. Consider a scheduling problem defined with one deadline $d = 170$, four machines ($m_1$ to $m_4$), the first two working with product $\pi_1$ and the others working with product $\pi_2$, setup times $(t_1, t_2) = (1, 3)$, demands $(q_1, q_2) = (74, 15)$, ready times $r = (150, 145, 135, 150)$, and a scheduling gap $[r_0, d_0] = [110, 124]$ assigned to machine $m_0 = m_2$. The algorithm produces scheduling with new ready times $r' = (170, 170, 170, 165)$, three machines working with product $\pi_1$ ($m_1$ to $m_3$), the old scheduling gap occupied with the processing of product $\pi_1$, and a setup time for product $\pi_1$ fixed at $r_0 = 110$. Moreover, a new scheduling gap $[r'_0, d'_0] = [135, 154]$ together with a setup of size $t_1 = 1$, is allocated on machine $m'_0 = m_3$. The top half of Figure 2 indicates the ready time for each machine and the lower half shows the tasks, with the allocation order indicated in parentheses. □

Figure 3 shows the sequence of setups supposedly allocated by successive iterations of the algorithm. Observe that this sequence tends to form waves along the time axis, hence the term wavy scheduling for this type of solution.

### Notation

In order to specify complete scheduling, a matrix $r_m^n$ indicating the ready time of machine $m$ at iteration $n$ of the algorithm is used; these variables

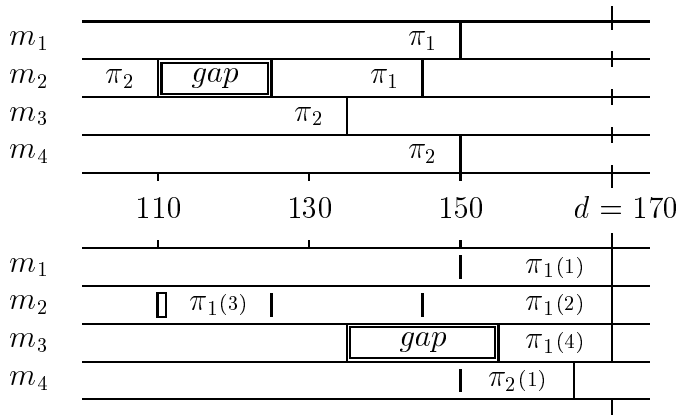


Figure 2: *Example of One-Deadline Scheduling*

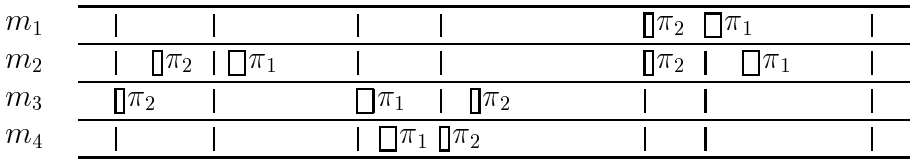


Figure 3: *Example of Wavy Scheduling*

represent the accumulated work allocated to the machines at the deadline $d_n$; $d_{n+1} - r_m^n$ is the time available on machine $m$ for the next iteration. In addition, $s_m^n$ is used to specify the start time of each setup. The basic notations are presented in Table 1.

| | | |
|---|---|---|
| Indices | | |
| $i \in \{1, 2\}$ | – | product index |
| $m \in \{1 \dots M\}$ | – | machine index |
| $n \in \{1 \dots N\}$ | – | order and iteration index |
| $\mu \in \mathcal{M}$ | – | gap index – $\mathcal{M} = \{ m \pm 0.1 : m = 1..M \}$ |
| Data | | |
| $t_i$ | – | setup time of product $\pi_i$ |
| $d, d_n$ | – | deadline of order $n$ |
| $q_i, q_i^n$ | – | processing time demanded by product $i$ in order $n$ |
| Variables | | |
| $r_m, r_m^n$ | – | ready time of machine $m$ at iteration $n$ |
| $s_m, s_m^n$ | – | start of the setup on machine $m$ at iteration $n$ |
| $[r_0, d_0], [r_0^n, d_0^n]$ | – | scheduling gap at iteration $n$ |
| $R = (r, [r_0, s_0], \mu)$ | – | ready time vector |
| $m_0 = round(\mu)$ | – | machine with scheduling gap |

Table 1: Basic Notations

**Sharp Ready Time Vector**

The algorithm works for the machines grouped according to the product currently being produced and ordered according to the ready times within the group. We assume that initial ready times satisfy $r_m \leq d_1, \ \forall m$ so that each machine starts with a nonnegative available processing time $d_1 - r_m$. Machines are sorted by order of decreasing ready times in group 1 and by increasing ready times in group 2. Such ordering is maintained throughout the execution of the algorithm. A non-integer value $\mu$ will thus be used to separate the machines according to group, with a machine with index $m < \mu$ belonging to group 1 and one with an index $m > u$ belonging to group 2. Notice that any initial ready times can be transformed to satisfy

these conditions by combining and eliminating scheduling gaps and reordering the machine indices. These properties give rise to a ready time vector $R = (r_1, \ldots, r_M, [r_0, d_0], \mu)$.

For each iteration, the current ready time vector $R$ is said to be *sharp* because it satisfies the following conditions:

- there is an index $\mu$ and at most one scheduling gap $[r_0, d_0]$
- machines with an index $m < \mu$ are working on product $\pi_1$ and the ready times of these machines form a non-increasing sequence: $r_m \leq r_{m-1}$, for $1 < m < \mu$
- machines with an index $m > \mu$ are working on product $\pi_2$ and the ready times of these machines form a non-decreasing sequence: $r_m \leq r_{m+1}$, for $\mu < m < M$

Each scheduling gap generated by the algorithm is characterized by a *gap index* $\mu$ and a net available processing time during the interval $[r_0, d_0]$ on machine $m_0$. The initiation of the setup in the scheduling gap is fixed later on in accordance with the jobs remaining to be allocated. Therefore, $d_0 + t < d = d_n$, where $t$ denotes the appropriate setup time: $t_1$ for product $\pi_1$ or $t_2$ for product $\pi_2$. In order to designate the machine with the scheduling gap, a specific set of values $\mathcal{M}$ is assigned to $\mu$. When a scheduling gap is created on machine $m_0$ which is working with product $\pi_1$ ($\pi_2$), the gap index is set to $\mu = m_0 + .1$ ($\mu = m_0 - .1$) and the setup involved will be for product $\pi_2$ ($\pi_1$). Therefore, $m_0 = round(\mu)$ and a machine $m$ is working with $\pi_1$ whenever $m < \mu$, and with $\pi_2$ whenever $m > \mu$.

**One-Deadline Problem**

A scheduling problem with $n$ deadlines is solved by solving a sequence of $n$ one-deadline problems. A *one-deadline problem* $P = P(d, q, R)$ is defined by a deadline $d$, processing time demands for both products $(q_1, q_2)$, and a sharp ready time vector $R = (r, [r_0, d_0], \mu)$ where $r = (r_1, \ldots, r_M)$. If the problem $P$ is feasible, the algorithm produces a scheduling which is a new ready time vector $R' = (r', [r'_0, d'_0], \mu')$ (characterized in the Appendix B) with the following properties:

- this vector is sharp

- there are three sets of adjacent machines, at least one of which must be non-empty:
  – machines working with $\pi_1$ without additional setup ($m < \mu, \mu'$)
  – machines for which additional setup was necessary (if $\mu \neq \mu'$)
  – machines working with $\pi_2$ without additional setup ($m > \mu, \mu'$)

- all new setups are for only one of the products ($\pi_1$ or $\pi_2$) and they form a non-decreasing sequence in time, whereas the sequence of machine indices will be either increasing (for $\pi_1$) or decreasing (for $\pi_2$)

In other words, such wavy scheduling is defined as a feasible scheduling of $P$ that requires time of exactly $q_1$ for product $\pi_1$ and $q_2$ for product $\pi_2$; *for every positive integer $k$, given the first $k$ setups with the last one allocated on a machine $m$, the next setup, if one exists, is allocated at the earliest possible time on the machine $m+1$ ($m-1$) whenever the setups are for product* $\pi_1$ ($\pi_2$). This earliest possible time can neither cause infeasibility nor increase the total number of setups. Clearly, if a setup for product $\pi_1$ ($\pi_2$) needs to be allocated, every machine working with this product is already occupied, and the scheduling gap has been eliminated; hence, the setup will be allocated to the machine with the earliest ready time (or greatest availability) of those that are currently producing $\pi_2$ ($\pi_1$). Therefore, every setup is allocated at the earliest possible ready time $s_m = r_m$ with the possible exception of the final one, when a scheduling gap $[r'_0, d'_0]$ may be created. In addition, Theorem 1 shows that wavy scheduling is an optimal scheduling.

**Wavy Scheduling**

Consider a scheduling problem $P_n$ with $n$ deadlines. The problem obtained from $P_n$ by considering just the first $n-1$ orders and deadlines is denoted by $P_{n-1}$. The *wavy scheduling* $R^n = (r^n, [r^n_0, d^n_o], \mu_n)$ of $P_n$ is defined as the scheduling produced with the one-deadline problem $P(d_n, q^n, R^{n-1})$. Therefore, $R^1, R^2 \ldots R^n$ denotes a sequence of wavy schedulings, representing a solution which is greedy in relation to deadlines. Moreover, wavy schedulings are extreme in the following sense:

**Wavy Scheduling Property**. *Given any first $k$ setups, with the last one allocated on machine $m$, the next, if there is one, is allocated at the earliest possible time on:*

- *the same machine $m$, if setups $k$ and $k+1$ are for opposite products*
- *machine $m+1$, if setups $k$ and $k+1$ are both for product $\pi_1$*
- *machine $m-1$, if setups $k$ and $k+1$ are both for product $\pi_2$*

**Theorem 1** *Wavy scheduling $R^n$ of a feasible problem $P_n$ is an optimal solution.*

**Proof**. Suppose that non-wavy $\bar{R}$ is an optimal scheduling for $P_n$. Using wavy scheduling transformation presented in appendix C, $\bar{R}$ is transformed into a wavy scheduling $R$. This maintains feasibility and the number of setups. Therefore, $R$ represents optimal scheduling. On the other hand, if $P_n$ does not admit wavy scheduling, then no scheduling is possible, i.e., Problem $P_n$ is infeasible.
*Observation.* The transformation is applied one deadline at time in order to maintain feasibility. □

**Theorem 2** . *Wavy scheduling simultaneously minimizes each one of the following objectives:*
- *the number of setups for each product,*
- *the total number of setups,*
- *the total time for setups of each product,*
- *the total time for setups,*
- *any non-decreasing function of the number of setups for each product.*

**Proof**. Theorem 1 can be used to prove the first objective concerning the number of setups of each product, because no setups are added during transformation. This means that the number of setups for each product will not increase, nor will the sum of any regular function involving the number of setups of each product. Since the other four objectives are regular functions, involving only the number of setups, they will also be minimized. □

### Complexity Analysis

**Theorem 3** . *If the scheduling problem is feasible, then the algorithm terminates with wavy scheduling; otherwise the problem is infeasible.*

**Proof**. If $P_N$ is feasible, then, by Theorem 1, it has wavy scheduling, and the algorithm constructs it. If the algorithm does not produce wavy scheduling for a problem $P_n$, $n \leq N$, then this problem is infeasible, which means that $P_N$ is infeasible. □

**Theorem 4** . *The algorithm is strongly polynomial with time complexity* $O(MN + M\log M + N\log N)$.

**Proof**. The algorithm constructs wavy scheduling with $N$ iterations of $O(M)$, for a total of $O(MN)$. However, if it is necessary to sort the $M$ machines initially to obtain sharp scheduling, $O(M\log M)$ should be added to the total time complexity. Moreover, if it is necessary to sort according to deadline demands, $O(N\log N)$ should be added to the total time complexity. Moreover, since the algorithm executes only operations of additions, subtractions, and comparisons, it is a strongly polynomial algorithm. □

# 3 Conclusions

Scheduling is a rich area for NP-hard problems, which are usually approached by heuristics. This paper presents a routine for lotsizing and scheduling jobs involving setups and deadlines on parallel machines which is useful in the design of such heuristics. The traditional strategy of balanced schedulings uses equal division of work among machines, whereas the algorithm suggested here provides wavy scheduling, which tends to be unbalanced (i.e., for each product, the end-of-scheduling of machines varies greatly).

The proof for the theorem of wavy scheduling assumes that the initial ready times satisfy $r_m \leq d$, $\forall m$. However, it is easy to modify the algorithm so that when $r_m > d$ for some machine $m$ then no allocation is made for this machine and its ready time remains the same: $r'_m = r_m$.

The algorithm is greedy in relation to deadlines. It is also possible to develop another version of the same algorithm which would be greedy as far as machines are concerned, making the same allocations as does the original, but in a different order. This leads to the belief that good heuristics to deal with various orders and machines can be developed for extensions of the model.

The results are due to the advantages of the model, including homogeneity from the use of identical machines and the bipolarity introduced by the restriction to two products. It has been suggested [14] that an analogy can be viewed between two-product and two-machine restrictions, but that the former is relatively less explored. The restriction to two products implies

difficulties in application, and the algorithms are not as flexible as when the number of machines is restricted. The latter situation leads to algorithms that are easily adapted to situations where more machines are involved. Nevertheless, looking at problems with two products can provide useful insights for problems where more products are involved. In actual practice, grouping various types of products as if they were a single one is indeed a common planning strategy.

There would also be no special difficulty involved in extending the algorithm to deal with issues of product availability (stock). Existing stock must be used to fulfill the most urgent orders, a situation which also applies to problems with various products. Stock should not be preserved, since it does not matter how many units of a product are in the inventory when another product is needed. Only if no existing stock is available should work be allocated to meet demands, first considering machines working the same product, then the filling of scheduling gaps, and finally machines working with the other product.

The algorithm produces a very suitable scheduling for situations where cancellations of orders, acceptance of new orders, and even machines breakdowns can occur. If an order of deadline $d_{\bar{n}}$ changes, there are no alterations on the partial schedulings of orders with deadlines $d_n < d_{\bar{n}}$ and they can be used to recompute the new optimal scheduling.

In addition, the wavy schedulings naturally minimize the number of machines needed. On the other hand, by interchanging allocations among machines it is possible to transform the wavy scheduling into a balanced optimal scheduling with a difference of at most 2 setups for any pair of machines.

# Appendices

### A Algorithm

The algorithm is presented in Table 2 using an algol-like language with comments between exclamation points (!). Each iteration of the algorithm starts with a sharp ready time vector and constructs wavy scheduling, which is used as a sharp ready time vector for the next iteration. Initial sharp ready times are assumed.

**Algorithm Wavy Scheduling**.
**start** with sharp ready time vector $(r, \mu)$ satisfying
$r_m \le d_1$, $\forall m$, and $r_0 < d_0 < d_1$
**for** ( $n \leftarrow 1..N$ ) **do** ! for every deadline !
$d \leftarrow d_n$; $q1 \leftarrow q_1^n$; $q2 \leftarrow q_2^n$
! assign $\delta$ units of $\pi_1$ on a group 1 machine $m$ !
**for** ($m \leftarrow 1..\lfloor \mu \rfloor$ **with** $q1 > 0$ ) **do**
$\delta \leftarrow \mathbf{min}\{d - r_m, q1\}$; $q1 \leftarrow q1 - \delta$; $r_m \leftarrow r_m + \delta$;
! assign $\delta$ units of $\pi_2$ on a group 2 machine $m$ !
**for** ( $m \leftarrow M..\lceil \mu \rceil$ **with** $q2 > 0$ ) **do**
$\delta \leftarrow \mathbf{min}\{d - r_m, q2\}$; $q2 \leftarrow q2 - \delta$; $r_m \leftarrow r_m + \delta$;
! assign $\delta$ of $\pi_1$ on the scheduling gap !
**if** ( $q1 > 0$ ) **then** $\delta \leftarrow \mathbf{min}\{d_0 - r_0, q1\}$; $q1 \leftarrow q1 - \delta$;
**if** ( $round(\mu) < \mu$ ) **then** $r_0 \leftarrow r_0 + \delta$; **else** $s^n_{round(\mu)}, d_0 \leftarrow d_0 - \delta$
! assign $\delta$ of $\pi_2$ on the scheduling gap !
**if** ( $q2 > 0$ ) **then** $\delta \leftarrow \mathbf{min}\{d_0 - r_0, q2\}$; $q2 \leftarrow q2 - \delta$;
**if** ( $round(\mu) > \mu$ ) **then** $r_0 \leftarrow r_0 + \delta$; **else** $s^n_{round(\mu)}, d_0 \leftarrow d_0 - \delta$
! assign setup and $\delta$ of $\pi_1$ on a group 2 machine $m$ !
**for** ( $m \leftarrow \lfloor \mu \rfloor..M$ **with** $q1 > 0$ ) **do**
**if** ( $t1 + q1 > d - r_m$ )
**then** $\delta \leftarrow d - r_m - t1$; $q1 \leftarrow q1 - \delta$; $s^n_m \leftarrow r_m$; $r_m \leftarrow r_m + \delta$
! create a new scheduling gap !
**else** $\mu \leftarrow m + .1$; $r_0 \leftarrow r_m$; $d_0 \leftarrow d - t1 - q1$; $q1 \leftarrow 0$;; $r_m \leftarrow d$;
! assign setup and $\delta$ of $\pi_2$ on a group 1 machine $m$ !
**for** ( $m \leftarrow \lceil \mu \rceil..1$ **with** $q2 > 0$ ) **do**
**if** ( $t2 + q2 > d - r_m$ )
**then** $\delta \leftarrow d - r_m - t2$; $q2 \leftarrow q2 - \delta$; $s^n_m \leftarrow r_m$; $r_m \leftarrow r_m + \delta$
! create a new scheduling gap !
**else** $\mu \leftarrow m - .1$; $r_0 \leftarrow r_m$; $d_0 \leftarrow d - t2 - q2$; $q2 \leftarrow 0$; $r_m \leftarrow d$;
**if** ( $q1 + q2 > 0$ ) **then stop** ($'problem\ is\ infeasible'$)

Table 2: Algorithm of Wavy Schedulings

## B Characterization of One-Deadline Wavy Schedulings

The algorithm is presented with an $M$-vector of current ready times $r_m$ instead of a matrix. Setup times, however, are maintained in the form of a matrix $(s_m^n)$. Although it would be possible to replace the matrix with two vectors, the second being a setup counter, this was avoided to maintain a constant notation. The largest (smallest) integer not larger (not smaller) than $\mu$ is designated as $\lfloor \mu \rfloor$ ($\lceil \mu \rceil$). The maximum quantity of allocable processing time $Q^0(j), Q^1(j), Q^2(j)$ obtained with $|j|$ setups is defined as $j$ setups of $\pi_1$ if $j$ is positive or $-j$ setups of $\pi_2$ if $j$ is negative. These times are computed recursively as follows:

| | |
|---|---|
| $j = 0$ | $Q^0(0) = r'_0$<br>$Q^1(0) = \sum_{m<\mu}(d - r_m)$<br>$Q^2(0) = \sum_{m>\mu}(d - r_m)$ |
| $j = 1 \ldots \lfloor M + 1 - \mu \rfloor$ | $Q^0(j) = (d - r_m - t_1)$<br>$Q^1(j) = Q^1(j-1) + Q^0(j-1)$<br>$Q^2(j) = Q^2(j-1) - (d - r_m),\ m = \lfloor \mu + j \rfloor$ |
| $j = -1 \ldots \lceil -\mu \rceil$ | $Q^0(j) = (d - r_m - t_2)$<br>$Q^1(j) = Q^1(j+1) - (d - r_m),\ m = \lceil \mu + j \rceil$<br>$Q^2(j) = Q^2(j+1) + Q^0(j+1)$ |

**Theorem 5** . *A one-deadline problem $P$ is feasible if and only if there is an integer $j \in (-\mu, M + 1 - \mu)$ such that*

$$q_1 \le Q^1(j) + Q^0(j) \qquad q_2 \le Q^2(j) + Q^0(j) \qquad q_1 + q_2 \le Q^1(j) + Q^2(j) + Q^0(j)$$

*The lowest value for $|j|$ among those values of $j$ that satisfy these conditions is the minimum number of setups that can be used in optimal scheduling.*

**Proof**. The expressions of $Q^i(j)$ represent wavy scheduling with $j$ setups. □

**Theorem 6** . *In a feasible one-deadline problem $P$, wavy scheduling $S' = S'(r', \mu')$ has the following assignments where $|j|$ is the number of setups fixed by the application of the previous theorem.*

| | | |
|---|---|---|
| $j < 0$ | $\mu' = \mu + j - 0.2,$ | $\mu > m_0$ |
| | $\mu' = \mu + j,$ | $\mu < m_0$ |
| $j = 0$ | $\mu' = \mu$ | |
| $j > 0$ | $\mu' = \mu + j,$ | $\mu > m_0$ |
| | $\mu' = \mu + j + 0.2,$ | $\mu < m_0$ |

| | |
|---|---|
| $1 \leq m < \mu, \mu'$ | $r'_m = \min\{d, r_m + \max\{0, q_1 - \sum_{\ell<m}(d - r_\ell)\}\}$ |
| $\min\{\mu, \mu'\} < m < \max\{\mu, \mu'\}$ | $r'_m = d$ |
| $\mu, \mu' < m \leq M$ | $r'_m = \min\{d, r_m + \max\{0, q_2 - \sum_{\ell<m}(d - r_\ell)\}\}$ |

| | |
|---|---|
| $\mu' = \mu$ | $r'_0 = \min\{d_0, r_0 + \max\{0, q_1 - \sum_{m<\mu}(d - r_m)\},\ \mu < m_0$<br>$r'_0 = \min\{d_0, r_0 + \max\{0, q_2 - \sum_{m>\mu}(d - r_m)\},\ \mu > m_0$<br>$d'_0 = \min\{r_0, d_0 - \max\{0, q_2 - \sum_{m>\mu}(d - r_m)\},\ \mu < m_0$<br>$d'_0 = \min\{r_0, d_0 - \max\{0, q_1 - \sum_{m<\mu}(d - r_m)\},\ \mu > m_0$ |
| $\mu' > \mu$ | $d'_0 = d - (\lvert j \rvert * t_1 + q_1 - (d_0 - r_0) - \sum_{m<\mu'}(d - r_m))$<br>$r'_0 = r_{m'_0} + \max\{0, q_2 - \sum_{m>\mu'}(d - r_m)\}$ |
| $\mu' < \mu$ | $d'_0 = d - (\lvert j \rvert * t_2 + q_2 - (d_0 - r_0) - \sum_{m>\mu'}(d - r_m))$<br>$r'_0 = r_{m'_0} + \max\{0, q_1 - \sum_{m<\mu'}(d - r_m)\}$ |

**Proof**. This follows from the allocations made by the wavy scheduling. □

**Theorem 7** *The wavy scheduling of a one deadline problem $P$ is characterized by the following:*

• if a machine is fully occupied with $\pi_1$ ($\pi_2$), then every machine with a smaller (larger) index is fully occupied with $\pi_1$ ($\pi_2$):
$r'_m = d \Rightarrow r'_{m-1} = d,$ for $1 < m < \mu$
$r'_m = d \Rightarrow r'_{m+1} = d,$ for $\mu < m < M$

• if a machine is not fully occupied with $\pi_1$ ($\pi_2$), then every other machine with a larger (smaller) index is not fully occupied with $\pi_1$ ($\pi_2$):
$r'_{m-1} < d \Rightarrow r'_m = r_m,$ for $1 < m < \mu$
$r'_{m+1} < d \Rightarrow r'_m = r_m,$ for $\mu < m < M$

• if there is a scheduling gap for work of $\pi_1$ ($\pi_2$), then the previous machine is fully occupied with $\pi_1$ ($\pi_2$):
$\mu' = \mu,\ d'_0 < d_0 \quad \Rightarrow \quad r_{m_0} = d$
$\mu' = \mu < m_0,\ r'_0 > r_0 \quad \Rightarrow \quad r_{m_0-1} = d$
$\mu' = \mu > m_0,\ r'_0 > r_0 \quad \Rightarrow \quad r_{m_0+1} = d$

• if the scheduling gap if totally filled and a new one is created with a setup of $\pi_1$ ($\pi_2$), then a setup of $\pi_1$ ($\pi_2$) is determined for each machine $m$ between $\mu, \mu'$, with each one fully occupied with $\pi_1$ ($\pi_2$); moreover, the setup is fixed at one of the extremes of the old scheduling gap.

$$\begin{array}{lll}
\mu' > \mu & \Rightarrow & s_m = r_m,\ r'_m = d,\ \text{for } \mu < m < \mu',\ r_{m'_0} \leq r'_0 < d'_0 + t_1 < d \\
\mu' < \mu & \Rightarrow & s_m = r_m,\ r'_m = d,\ \text{for } \mu' < m < \mu,\ r_{m'_0} \leq r'_0 < d'_0 + t_2 < d \\
\mu' > \mu > m_0 \text{ or } \mu' < \mu < m_0 & \Rightarrow & s_{m_0} = r_0 \\
\mu' > \mu < m_0 \text{ or } \mu' < \mu > m_0 & \Rightarrow & s_{m_0} = d_0
\end{array}$$

**Proof**. This follows from the allocations made by the wavy scheduling. □

## C Wavy Scheduling Transformation

.

These routines can be used to transform a feasible scheduling $\bar{R}$ of a scheduling problem $P_n$ into wavy scheduling $\hat{R}$ of the scheduling problem $\hat{P}_n$ obtained from $P_n$, but with the demands of the final deadline $d_n$ increased up to the level of $\bar{R}$. The final routine eliminates production excesses, yielding wavy scheduling $R$ of $P_n$.

*Idle Gap Elimination.* For every idle gap (i.e., time interval $[s, t]$ in which a machine is not in use, although it is producing the same product both before and after the gap), the work scheduled at time $t$ is moved forward to time $s$ (Figure 4). At this point, $\bar{R}$ has no idle gaps.

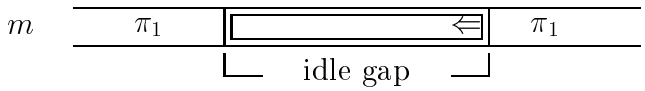


Figure 4: *Idle Gap Elimination*

*Extra Scheduling Gap Elimination.* For every pair of scheduling gaps $\gamma_1, \gamma_2$ where the end of $\gamma_1$ comes prior to the end of $\gamma_2$, the work scheduled after $\gamma_2$ is transferred to $\gamma_1$ until either $\gamma_1$ or $\gamma_2$ disappears; the gap $\gamma_1$ should be filled from left to right or vice-versa, depending on the product which is involved in the work transferred from $\gamma_2$. Making this transfer will guarantee that at least one of the scheduling gaps will disappear. If $\gamma_2$ increases and fuses with another setup, then two setups can be eliminated. If $\gamma_2$ increases until no more work remains to be allocated, the setup of $\gamma_2$ can be eliminated. If $\gamma_1$ decreases until no more work can be transferred, it is eliminated. In all

cases, the new scheduling is feasible and no worse than the old one. This reasoning can be applied for any pair of gaps. The final scheduling will have at most a single gap (see Figure 5). At this point, there is at most a single scheduling gap.

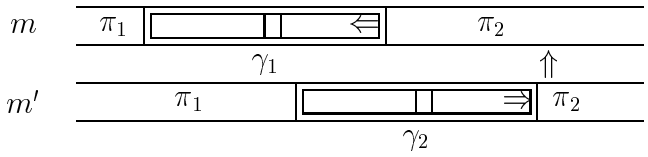


Figure 5: *Extra Scheduling Gap Elimination*

*Opposite Setup Delay.* For every pair of opposite setups in a given deadline interval $[d_{n-1}, d_n]$, whether on the same machine or on different machines, both are delayed by interchanging the ensuing work until one of the setups *a)* starts at the deadline $d_n$, b) fuses with another setup so that both can be eliminated, or c) has no more ensuing work so the final setup can be eliminated (Figure 6). At this point, all setups between two consecutive deadlines are of the same type.

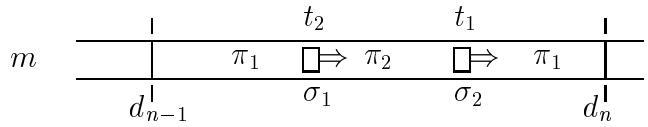


Figure 6: *Opposite Setup Delay*

*Scheduling Switch.* Let $\sigma$ be the first setup of $\pi_1$ ($\pi_2$) starting on a machine $m$ simultaneously with the production of the other product on the machine $m' < m$ ($m' > m$). For every such a setup, switch the work after the initial $\sigma$ from one machine to the other so that the ensuing production of $\pi_1$ ($\pi_2$) will be on machine $m$, and that of $\pi_2$ ($\pi_1$) on $m'$ (Figure 7). At this point, every sequence of adjacent setups of product $\pi_1$ ($\pi_2$) forms an increasing (decreasing) sequence of machine indices.

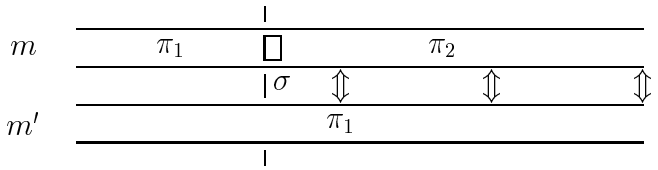


Figure 7: *Scheduling Switch*

*Setup Adjustment.* Let $\sigma_1$ be the first setup of $\bar{R}$ which is postponed with respect to the associated setup $\hat{R}^n_m$ of $\hat{R}$; this results in a lack of production of the product during setup and implies the existence of a setup of the same (opposite) product which has been moved forward (postponed). Let $\sigma_2$ be the first of such setups which is associated with $\hat{s}^{n'}_{m'}$. For each pair of such setups, $\sigma_1$ is moved forward, and $\sigma_2$ is delayed (moved forward) by transferring work of both products back and forth from one machine to the other until one of the setups matches on $\bar{R}, \hat{R}$. Feasibility is maintained because work moved forward never causes infeasibility and delay transfers work to a position prior to the deadline $d_{n'}$ which gave rise to the original allocation of the setup $\sigma_2$. At this point, the schedulings $\bar{R}$, $\hat{R}$ match.
*Observation.* The objective of the setup adjustment routine is to force the scheduling $\hat{R}$ to satisfy the wavy scheduling property which requires setups to be allocated at the earliest possible time. Although the routine is presented in terms of setups of scheduling $\hat{R}^n_m$, it is not necessary to assume the existence of wavy scheduling $\hat{R}$. Setups are allocated, one by one, for each deadline, in accordance with the formulae in the appendix B; each setup is fixed, with the possible exception of the one associated with the current scheduling gap, which depends upon the work demands of later deadlines. Even in this case, the setup is determined prior to the allocation of any new setup. Therefore, setups are examined one at a time, and work is transferred in order to move each setup examined forward as much as possible without losing feasibility with respect to final deadlines. Once a setup has been examined, it is fixed for the execution of the rest of the routine.

*Elimination of Excess Production.* Eliminate the surplus production of $\bar{R}$ with respect to $P_n$ by applying the rules of construction of wavy scheduling in the reverse order (Figure 9). At this point, the schedulings $\bar{R}$, $R$ match.

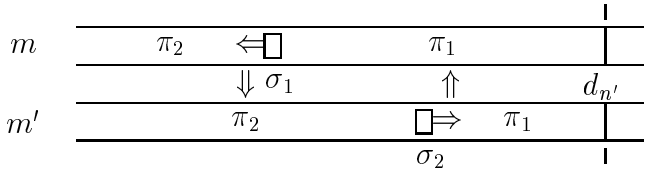


Figure 8: *Setup Adjustment*

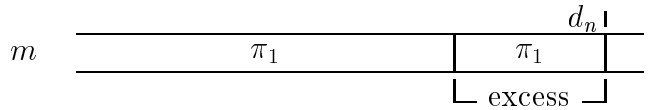


Figure 9: *Elimination of Production Excess*

# References


[1] Ahn, B. H., and Hyun, J. H., "Single Facility Multi-Class Job Scheduling", *Computers and Operations Research* **17/3** (1990) 265-272.

[2] Cattrysse, D., Salomon, M., Kuik, R., and Van Wassenhove, L. N., "A Dual Ascent and Column Generation Heuristic for the Discrete Lotsizing and Scheduling Problem with Setup Times", *Management Science* **39/4** (1993) 477-486.

[3] Cheng, T. C. E., and Sin, C. C. S., "A State-of-the-Art Review of Parallel Machine Scheduling Research", *European Journal of Operational Research* **47/3** (1990) 271-292.

[4] Dogramaci, A., and Surkis, J., "Evaluation of a Heuristic for Scheduling Independent Jobs on Parallel Identical Processors", *Management Science* **25/12** (1979) 1209-1216.

[5] Gonçalves, J. F., and Marques, M. P., "Heurística para o Sequenciamento de Lotes Económicos de $n$ Produtos em $m$ Máquinas Idênticas", *Investigação Operacional* **9/2** (1989) 47-61.

[6] Guinet, A., "Textile Production Systems: a Succession of Non-identical Parallel Processor Shops", *Journal of Operational Research Society* **42/8** (1991) 655-671.

[7] Gupta, J. N. D., "Optimal Schedules for Single Facility with Two Job Classes", *Computers and Operations Research* **11/4** (1984) 409-413. (See [11]).

[8] Gupta, J. N. D., "Single Facility Scheduling with Multiple Job Classes", *European Journal of Operational Research* **33/1** (1988) 42-45.

[9] Ho, J. C., and Chang, Y.-L., "Heuristics for Minimizing Mean Tardiness for m Parallel Machines", *Naval Research Logistics* **38/3** (1991) 367-381.

[10] Lee, C. Y., and Danusaputro, S., "Economic Lot Scheduling for Two-Product Problem", *IIE Transactions* **21/2** (1989) 162-169.

[11] Mason, A. J., and Anderson, E. J., "Minimizing Flow Time on a Single Machine with Job Classes and Setup Times", *Naval Research Logistics* **38/3** (1991) 333-350.

[12] Monma, C. L., and Potts, C. N., "On the Complexity of Scheduling with Batch Setup Times", *Operations Research* **37/5** (1989) 798-804.

[13] Olhager, J., and Rapp, B., "Setup Reduction in Production Systems: Review and Extensions", *Operational Research'87*, IFORS, (1988) 259-275.

[14] Sahney, V. K., "Single-Server, Two-Machine Sequencing with Switching Time", *Operations Research*, **20/1** (1972) 24-36. Errata in: 22/5 (1974) 1120.

[15] Salomon, M, Kroon, L. G., Kuik, R., and Van Wassenhove, L. N., "Some Extensions of the Discrete Lotsizing and Scheduling Problem", *Management Science* **37/7** (1971) 801-812.

[16] Serafini, P., and Speranza, M. G., "Production Scheduling Problems in a Textile Industry", *European Journal of Operational Research* **58/2** (1992) 173-190.

[17] So, K. C., "Some Heuristics for Scheduling Jobs on Parallel Machines with Setups", *Management Science* **36/4** (1990) 467-475.

[18] Unal, A. T., and Kiran, A. S., "Batch Sequencing", IIE Transactions, **24/4** (1992) 73-83.

[19] Webster, S., and Baker, K. R., "Scheduling Groups of Jobs on a Single Machine", *Operations Research* **43/4** (1995) 692-703.