RAIRO Operations Research
RAIRO Oper. Res. **41** (2007) 141–154
DOI: 10.1051/ro:2007016

SCHEDULING WITH PERIODIC AVAILABILITY CONSTRAINTS AND IRREGULAR COST FUNCTIONS

Francis Sourd[1]

Abstract. This paper addresses a one-machine scheduling problem in which the efficiency of the machine is not constant, that is the duration of a task is longer in badly efficient time periods. Each task has an irregular completion cost. Under the assumption that the efficiency constraints are time-periodic, we show that the special case where the sequence is fixed can be solved in polynomial time. The general case is NP-complete so that we propose a two-phase heuristic to find good solutions. Our approach is tested on problems with earliness-tardiness costs.

Keywords. Scheduling, earliness-tardiness, availability, break.

Mathematics Subject Classification. 90B35

1. Introduction

A library or a software to solve scheduling problems must offer the two following features: a language or an interface to describe the problem and a solver that finds a good feasible – ideally optimal – solution. The modelling facilities are the "visible part of the iceberg" for the user. In this way, a scheduling piece of software can be used by a person who does not know the whole scheduling theory. However, as the user does not usually know – and, often, does not want to know – the intractability of scheduling, he or she is mainly interested in modelling the problem as faithfully as possible. Therefore, someone who wants to buy a scheduling software will reject any product that is not able to model his or her problem even before comparing the solving performances. As a consequence, scheduling libraries must offer the ability to model a large variety of constraints even those that have been neglected

Received September 30, 2005. Accepted June 10, 2006.

[1] CNRS-LIP6. 4, place Jussieu 75252 Paris Cedex 05, France; Francis.Sourd@lip6.fr

by theorists because they either make the problems much too complex or are considered as non-fondamental.

This work is motivated by solving some of the instances of MascLib [10]. This library, proposed by ILOG researchers, contains instances that are close to some real problems. More precisely, we are interested in instances with periodic *efficiency constraints* and/or *break constraints*. These families of constraints, implemented in ILOG Scheduler 6.0 [8], model that the *duration* of activities may be longer when some resources are – totally or partially – unavailable. For any task J_i, the efficiency constraint can be formulated as follows:

$$\int_{S_i}^{C_i} \mathrm{eff}_i(t)\mathrm{d}t = p_i \tag{1}$$

where

- S_i and C_i are respectively the start time and the completion time of J_i. Both are decision variables of the problem;
- p_i is the given *processing time* of the problem;
- $\mathrm{eff}_i(t)$ is the *efficiency level* of J_i at time t. It is a given *piecewise constant* function from $\mathbb{R}^+$ to $[0,1]$.

Let us first observe that this definition slightly differs from the definition of ILOG Scheduler because ILOG Scheduler requires that these variables are integer in order to conform to the constraint programming engine. Because of the integrity of S_i and C_i, Equation (1) must be approximated.

As $\mathrm{eff}_i(t) \leq 1$, we have $C_i - S_i \geq p_i$, which means that we make a clear distinction between the duration of the task equal to $C_i - S_i$ and its processing time p_i, which relates to the normalized quantity of work necessary to complete the task. Indeed, when the efficiency is equal to 1, we have $C_i - S_i = p_i$. We also note that two tasks can be given two different efficiency levels.

When $\mathrm{eff}_i(t) = 0$ for all t in a time interval (s,e), we say that (s,e) is a *break period* for J_i. In scheduling theory, break periods are also known as *unavailability* periods [9]. Note that the unavailability periods are usually associated to a machine while, in our model, they are attached to the tasks. Attaching the unavailability periods to the tasks is useful, for example when some tasks must be interrupted during the night while others can be processed without any control. We also observe that when the efficiency levels are all equal ($\mathrm{eff}_i = \mathrm{eff}$) and $\mathrm{eff}(t) \in \{0,1\}$ for all t, the problem corresponds to the problem with unavailability periods and resumable tasks.

In practice, several types of constraints can be related to the presence of a break: either a task can be interrupted by the break and resumed at the end of the break or, conversely, it must be entirely scheduled in between a pair of consecutive break periods. Moreover, in some situations, a task can complete but cannot start during a break because a human operator is required to setup the task. It can even be required that a task starts at least x hours before a break. ILOG Scheduler API offers all the modelling facilities. In Section 2, we will show how all these efficiency and break constraints can be represented by a simple and unified data structure.

In this paper, we address the one-machine problem, which means that for any pair of tasks $J_i \neq J_j$, we must have either $C_i \leq S_j$ or $C_j \leq S_i$ (*disjunctive constraints*). The problem is to minimize the weighted sum of earliness and tardiness, which is the most general optimization criterion proposed in MascLib. The problem is clearly NP-hard, even if there is neither efficiency nor availability constraints. It is also in NP (we can check in polynomial time whether (1) is satisfied or not) so that the problem is NP-complete.

Our study is motivated by the fact that, in the instances of MascLib, the efficiency and breaks are usually periodic: the period is typically the day or the week. Therefore, the breaks and efficiency levels are compactly encoded: the input gives the information on a single period. From a theoretical point of view, problems with a compactly encoded input are often very difficult because they generate a pseudopolynomial number of constraints or variables (in our case, we will see that the efficiency curves have a pseudopolynomial number of levels). Therefore, finding a polynomial algorithm is usually difficult. Other problems with compactly encoded input typically appear in cyclic scheduling [5] and in high multiplicity scheduling [1, 2, 7].

The theoretical contribution of this article is related to the *timing* subproblem, which consists of finding the optimal start and completion times once the tasks are sequenced. We show that the problem can be solved in polynomial time by adapting an algorithm proposed by Sourd [11]. To this end, specific data structures are introduced in order to keep a compact representation of the dynamic programming states. Our second contribution is a local search algorithm to find good solutions for the general problem. This algorithm emphasizes the importance of having an efficient timing procedure since it is called for each candidate sequence. However, we show that an even more efficient algorithm can be derived by a two-phase algorithm that first solves a simplified approximate model and then works with the exact model.

In Section 2, the problem is formally defined and some modelling issues are addressed. Section 3 is devoted to the timing problem and we prove that it can be solved in polynomial time when the efficiency is periodic. This theoretical result is then applied in Section 4 to tackle the sequencing problem with a local search procedure and the benefits of our new approach are illustrated by experimental tests.

2. Definitions and model

2.1. Problem definition

A set of n tasks $J_1, \ldots, J_n$ is to be scheduled on a single machine. Each task is given a processing time p_i, an efficiency level eff_i and a cost function $f_i(C_i)$ which is a continuous piecewise linear function. For example, in the earliness-tardiness case, we have $f_i(C_i) = \max(\alpha_i(d_i - C_i), \beta_i(C_i - d_i))$. A feasible schedule is defined by its start and completion times $(S_i, C_i)_{1 \leq i \leq n}$ which must satisfy the efficiency constraints (1) and the disjunctive constraints formulated in the introduction. The problem is to minimize the total cost $\sum_i f_i(C_i)$.

In a first step, we do not consider the specific constraints about break periods (that is breaks are resumable). They will be addressed at the end of this subsection. In order to ensure there exists a feasible schedule, we assume that $\int_0^\infty \text{eff}_i = \infty$ and, to avoid that a task should be scheduled infinitely late, we assume that the slope of the latest segment of each cost function f_i is strictly positive (in fact, we can assume that they are non-negative but the proof of Lemma 1 would be longer).

For a given start time S_i, the possible completion times subject to (1) are clearly in an interval. If the function eff_i is strictly positive then we have only one possible completion time denoted by $C_i(S_i)$. Clearly, the function $S_i \mapsto C_i(S_i)$ is strictly increasing. The reverse is denoted by $C_i \mapsto S_i(C_i)$. Both functions are piecewise linear.

The rest of our study will intensively use this explicit relation between start and completion times. In order to have simpler proofs in Section 3, we are going to make the assumption that we have a bijection between start and end times, which is not true when there are break periods. However, we show that this assumption is not too restrictive.

Definition 1. For an instance I of the problem, we define the *ϵ-positive* instance I_ϵ by changing each efficiency level eff_i into the function $t \mapsto \max(\epsilon, \text{eff}_i(t))$.

Let us denote by $\text{OPT}(I)$ the optimal cost for the instance I. The following lemma shows that for small ϵ, I_ϵ is usually a good approximation of I.

Lemma 1. *For any instance I, we have*

$$\lim_{\epsilon \to 0^+} \text{OPT}(I_\epsilon) = \text{OPT}(I)$$

Proof. Let us consider a sequence $(\epsilon_k)_{k \geq 0}$ that converges to 0. We are going to prove that the sequence $\text{OPT}(I_{\epsilon_k})$ converges to $\text{OPT}(I)$. For each instance I_{ϵ_k}, let us consider an optimal solution $(S_i^k, C_i^k)_{1 \leq i \leq n}$. Clearly, we have for any i and k that $f_i(C_i^k) \leq \text{OPT}(I_{\epsilon_k}) \leq \text{OPT}(I)$ so that there exists a time *horizon* H (independent of k) such that $C_i^k \leq H$. For each k, the vector $(S_i^k, C_i^k)_{1 \leq i \leq n}$ is in $[0, H]^{2n}$ which is a compact. Therefore, we can extract a sequence of schedules that converge to a schedule denoted by $(S_i^\star, C_i^\star)_{1 \leq i \leq n}$. Since $\text{OPT}(I_{\epsilon_k}) \leq \text{OPT}(I)$ for all k, the cost of this schedule is less than or equal to $\text{OPT}(I)$. Furthermore, by continuity, the schedule $(S_i^\star, C_i^\star)_{1 \leq i \leq n}$ satisfies all the constraints so that it is feasible and therefore optimal. □

This lemma justifies that we can work with an ϵ-positive instance and therefore with bijective $S_i(C_i)$ and $C_i(S_i)$ functions. But what happens if there are additional constraints related to the break periods?

- **A task cannot end during a break**. Clearly, in the proof of Lemma 1, the completion time $C_i^\star$ of some task J_i can be inside a break period (s, e). Therefore I_ϵ is no more a good approximation. However, if there is no strict local minimum of f_i inside the interval (s, e), we can easily adapt the schedule $(S_i^\star, C_i^\star)$ so that no task starts or ends during a break. In practice, this assumption is not restrictive: it simply means that no due date must be planned during the nights or the week-ends.

- **A task cannot be interrupted by a break**. If the task J_i must be wholly processed before or after a break (s, e), then it cannot start in the interval $(S_i(s), s)$. Therefore, we can modify the function $C_i(S_i)$ such that, for any $t \in (S_i(s), e)$, we set $C_i(t)$ to be equal to the value $C_i(e)$. Consequently, the task can be processed just after the break in the time interval $[e, C_i(e))$. However, we observe that the modified function $t \mapsto C_i(t)$ is no more strictly increasing but we can transform the constant segment on the interval $(S_i(s), e)$ into a segment with a very small slope. Similarly to Lemma 1, we can then prove that this approximation is good when the slope tends to zero.

Following these observations, we will consider in the rest of this paper that we are given the two functions $C_i(S_i)$ and $S_i(C_i)$ instead of the efficiency levels eff_i (and the eventual additional constraints). From an algorithmic point of view, both functions $C_i(S_i)$ and $S_i(C_i)$ can be easily derived from the function eff_i in linear time of the number of different efficiency levels in eff_i. Before going further, we introduce the issue of the periodic efficiency.

2.2. Periodic efficiency

From now on, we consider that for each task the efficiency levels are periodic of period T. The breaks are then also periodic. Therefore, we clearly have $C_i(t+T) = C_i(t) + T$ for any t. According to the following definition, $C_i(t)$ will be said to be (T, T)-periodic.

Definition 2. A function $f : D_f \to \mathbb{R}$ (D_f is an interval of $\mathbb{R}$) is (T, δ)-*periodic* if and only if $f(t + T) = f(t) + \delta$ for all t such that t and $t + T$ are in D_f. In other words, the function $t \mapsto f(t) - (\delta/T)t$ is T-periodic.

We have mentioned in the introduction that a periodic efficiency level can be compactly encoded since we only need it to be represented over $[0, T)$. Similarly, $S_i(C_i)$ and $C_i(S_i)$ can also be compactly encoded.

3. Timing problem

In this section, we consider the timing problem for a given sequence, that is we add to our problem the constraints $C_i \leq S_{i+1}$ for all $i \in \{1, \ldots, n-1\}$. We immediately note that this problem can be solved by the dynamic programming approach of Sourd [11] but it requires to have an explicit representation of the function $S_i(C_i)$ so that the resulting time complexity, which depends on the number of segments of $S_i(C_i)$, is pseudopolynomial. Therefore, our work is to adapt the algorithm so that it becomes polynomial. We also note that none of the many approaches derived from the algorithm of Garey et al. [3] can be adapted because they rely on a linear programming formulation which would require $S_i(C_i)$ to be convex — it is not generally the case unless eff_i is constant.

3.1. Dynamic programming scheme

The dynamic programming approach is based on the following equation

$$\Sigma_k(t) = \begin{cases} f_1(t) & \text{for } k = 1 \\ \min_{\theta \le S_k(t)} \Sigma_{k-1}(\theta) + f_k(t) & \text{for } k > 1 \end{cases} \tag{2}$$

where $\Sigma_k(t)$ is the minimal cost to schedule the tasks $J_1, \ldots, J_k$ in the time interval $[0, t]$ subject to the constraint $C_k = t$ and $f_k(t)$ is the cost of J_k when it completes at t. The idea in [11] is to prove that all the states of the dynamic program can be compactly encoded by piecewise linear functions: indeed, the number of segments of these functions stays polynomially bounded. Due to the periodicity, it is no more the case but, as observed at the end of Section 2.2, functions $S_k(t)$ can be compactly encoded and we are going to extend this compact encoding to the functions Σ_k — which are *not* (T, δ)-periodic.

3.2. Polynomial algorithm

We will say that the encoding of $S_k(t)$ is a T*-metasegment* (or simply a metasegment as T is a constant). More precisely, this data structure contains the definition domain $[s, e)$ (with $s \in \mathbb{R}^+$, $e \in \mathbb{R}^+ \cup \{\infty\}$) and the description of the function over the *support interval* $[s, \min(e, s + T))$, this description is called the *support* of the metasegment. Since the function is piecewise linear, the support is the (x-ordered) list of the coordinates of the breakpoints of the piecewise linear function over the support interval. If $s - e > T$, the metasegment if said to be periodic, otherwise it is non-periodic.

We will say that a function is *meta-piecewise linear* if it can be decomposed into a list of metasegments, that is its domain definition can be partitioned into a finite number of intervals such that the restriction of the function over each interval is a metasegment. Clearly, there may be many different ways to represent a piecewise linear function f as a meta-piecewise linear function. In the worst case, each segment of f could be represented by a metasegment, which would mean that period patterns are all ignored. However, in this algorithm, we will focus on keeping the size of the data structure polynomial. Indeed, the space complexity of the representation of a meta-piecewise linear function f depends not only on the number of metasegments but also on the complexity of each metasegment. For simplicity, we will only consider the number $N(f)$ of metasegments and the maximal number of segments in all the metasegment supports, denoted by $M(f)$. For the proof, we will separate the non-periodic and periodic metasegments, the number of which is respectively denoted by $N_1(f)$ and $N_2(f)$. Clearly $N(f) = N_1(f) + N_2(f)$. The space complexity of a meta-piecewise linear function f is then in $O(N(f)M(f))$.

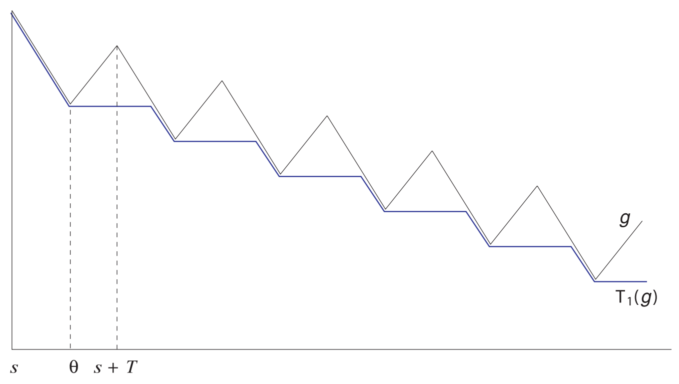

FIGURE 1. $T_1(f)$ for a (T, δ)-periodic function.

We have seen that any piecewise linear function is also meta-piecewise linear, thus all the functions $\Sigma_k(t)$ are meta-piecewise linear. The difficulty is to show that $N(\Sigma_k)$ and $M(\Sigma_k)$ are polynomial. In order to derive Σ_k from Σ_{k-1}, Equation (2) indicates that the three following operations must be performed, that is $\Sigma_k = T_3^{f_k}(T_2^{S_k}(T_1(\Sigma_{k-1})))$ with

(1) $T_1(f) : f \mapsto (t \mapsto \min_{\theta \le t} f(\theta))$;
(2) the composition $T_2^{S_k}(f) = f \circ S_k$;
(3) the addition $T_3^{f_k}(f) = f + f_k$.

We will denote by $\|f_k\|$ the number of segments of the piecewise linear function f_k and by $\|S_k\|_T$ the number of segments of S_k over its support interval. We first give three lemmas that indicate how the complexity of a meta-piecewise linear function is modified by applying each of the three transformation. The proofs are quite technical but intuitive so that we deliberately give informal proofs. The reference to the non-periodic case [11] is also very useful.

Lemma 2. *For any meta-piecewise linear function f, $T_1(f)$ is meta-piecewise linear and we have the three following inequalities: $N_2(T_1(f)) \le N_2(f)$, $N_1(T_1(f)) \le N_1(f) + N_2(f)$ and $M(T_1(f)) \le M(f)$.*

Proof. Let us first consider the piecewise linear representation of f. From [11], $T_1(f)$ has less segments than f because each segment is either left unchanged or transformed into a horizontal segment. However, as illustrated by Figure 1, a difficulty arises when dealing with a metasegment: a (T, δ)-periodic function g is not transformed into a (T, δ)-periodic function. However, if $[s, e)$ denotes the definition demain of g, $T_1(g)$ is clearly (T, δ)-periodic over $[\theta, e)$ for some $\theta \in [s, s+T)$. Therefore, $T_1(g)$ can be represented by a non-periodic metasegment over (s, θ) and a metasegment over $[\theta, e)$. From [11], we can prove that the support of each of the two metasegments is no more complex than the support of g (if θ

is well chosen, in order to correspond to a breakpoint) and the number of non-periodic segments over $[s, \theta)$ is clearly less than $M(g)$. Finally, we prove the lemma by applying a similar upper bounding to each metasegment of f. □

Lemma 3. *For any meta-piecewise linear function f and a (T, T)-periodic function S_k, $T_2^{S_k}(f)$ is a meta-piecewise linear function such that $N_1(T_2^{S_k}(f)) \leq N_1(f)$, $N_2(T_2^{S_k}(f)) \leq N_2(f)$ and $M(T_2^{S_k}(f)) \leq M(f) + \|S_k\|_T$.*

Proof. Let us consider a – periodic or non-periodic – metasegment over the interval $[s, e)$. $T_2^{S_k}$ transforms this metasegment into a metasegment over $[S_k^{-1}(s), S_k^{-1}(e))$. From [11] and since the length of the support is less than a period, we know that at most $\|S_k\|_T$ breakpoints are added to the support of the metasegment, whether it is periodic or not. □

Lemma 4. *For any meta-piecewise linear function f and a piecewise linear function f_k, $T_3^{f_k}(f)$ is a meta-piecewise linear function such that $N(T_3^{f_k}(f)) \leq N(f) + \|f_k\|$ and $M(T_3^{f_k}(f)) \leq M(f)$.*

Proof. We cut the metasegments of f at each breakpoint of f_k, which increases the number of metasegments by at most $\|f_k\|$. A linear function is added to the support of each metasegment, which means that the complexity of the supports is unchanged. □

Let us now combine these three lemmas. Since the complexity of the support is only changed by the operators $T_2^{S_k}$, we clearly have that $M(\Sigma_k) = O(\sum_i \|S_i\|_T)$. The periodic metasegments are only created by $T_3^{f_k}$ so that $N_2(\Sigma_k) = O(\sum_i \|f_i\|)$. Since T_1 is applied n times, we have $N_1(\Sigma_k) = O(n \sum_i \|f_i\|)$. We have then proved the complexity of the representation of the functions Σ_k and the corollary regarding the complexity of the dynamic programming algorithm.

Theorem 5. *For any k, Σ_k is a meta-piecewise linear function with $N(\Sigma_k) = O(n \sum_i \|f_i\|)$ and $M(\Sigma_k) = O(\sum_i \|S_i\|_T)$.*

Corollary 6. *The timing problem is polynomial and it can be solved in $O(n^2 \sum_i \|f_i\| + n \sum_i \|S_i\|_T)$ time.*

3.3. Experimental tests

Table 1 compares two implementations in C++ of the timing algorithm to solve instances from MascLib. In the "pseudopolynomial implementation", the cost functions of the dynamic programming scheme are explicitely encoded, which means that they have a pseudopolynomial number of segments: it is a direct implementation of [11]. The "improved implementation" is based on the polynomial algorithm presented in this section.

As the timing procedure is very fast, CPU times presented in the table do not correspond to a single timing but to a complete local search descent which calls the timing algorithm for each candidate neighbor (the neighborhood is defined in Sect. 4). Of course, we compare two descents from the same initial sequence in

TABLE 1. Improvement of the timing algorithm with the metasegments.

Instance	Number of tasks	Pseudopolynomial implementation	Improved implementation
BROS-15	30	1.35s	0.13s
BROS-15a	30	0.80s	0.25s
BROS-41	90	1304s	88.2s
BROS-41a	90	531s	91.9s
STC-BROS-15	30	9.01s	0.26s
STC-BROS-15a	30	11.3s	0.29s
STC-BROS-41	90	1011s	197s
STC-BROS-41a	90	1020s	202s

order to have the same number of call to the timing procedure. The improvement of our new algorithm is clear. However, we observe that a descent is quite time consuming for instances with 90 tasks, even with the improved timing procedure.

4. Local search

This section is devoted to solving the general scheduling (sequencing and timing) problem. The approach is an iterated local search procedure, where, at each iteration, the local search procedure starts from a random sequence. This method was shown to be very efficient for the basic problem without efficiency constraints [12] since the mean deviation is about 0.1% when the local search is run n times (n is the number of tasks in the instance). In particular, the choice of starting each descent with a random sequence comes from the fact that there is no good and fast heuristic for the earliness-tardiness problems.

The main drawback illustrated by the experimental results presented in Section 3.3 is that the CPU time for a single descent may be large. In a practical approach, the CPU time is usually limited so that only a small number of descents can be run. Therefore, the heuristic search may return a bad local optimum — we will indicate in Section 4.2 how bad the local optima may be.

4.1. Using an approximate instance

Figure 2 shows a metasegment of a function Σ_k obtained when solving the timing problem for a typical instance. On the one hand, we observe that the metasegment is indeed (T, δ)-periodic, which means that the data structure described in the previous section is appropriate. On the other hand, the breakpoints of the metasegment are nearly aligned so that we would like to approximate the metasegment by a single segment. Indeed, in the instances we have to solve, we observe that the due dates are distant by several dozens of days while the variance of the task duration is about a couple of days.

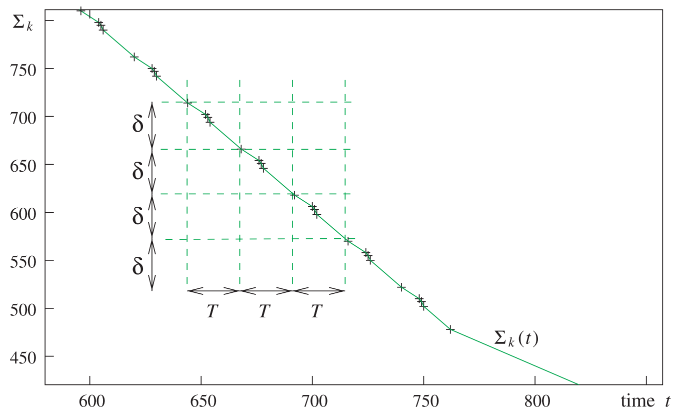

FIGURE 2. The meta-piecewise linear function Σ_k.

Based on this observation, the idea for a faster local search algorithm is to define a simpler, *approximate* instance and to search for good sequences for the approximate instance. Then, these good sequences are used as input for a local search algorithm for the initial *exact* problem. Using a simpler model helps to quickly find good sequences and, since the second phase is initialized with good solutions the number of calls to the complex timing procedure of Section 3 is greatly reduced.

The approximate model is defined by removing the efficiency constraint and by replacing each processing time by the mean duration of the task in the initial model, that is

$$p_i^{\text{apx}} = p_i \frac{T}{\int_0^T \text{eff}_i(t)\text{d}t} \cdot \tag{3}$$

Clearly, the approximate instance is an instance of the basic single machine problem with earliness and tardiness penalties. As mentioned in the beginning of the section, local search finds good solutions for this problem and the neighborhood search can even be accelerated using *ad hoc* data structures [6].

The local search is identical for both the approximate and the exact models. A solution is represented by the task sequence. The neighborhood of a solution σ is the set of sequences obtained by applying the generalized pairwise interchange operator (see *e.g.* [4]) which is the combination of the two following operators:

- swap two tasks in the sequence;
- extract a task and reinsert it to another position in the sequence.

The cost of each candidate sequence is evaluated by calling the timing algorithm (the classic one for the approximate model or the one presented in Section 3 for the exact model).

We now give a more precise description of the two phases of the algorithm.

1. A collection $\mathcal{C}$ of good sequences for the approximate instance is built by running the local search procedure with different random initial sequences.

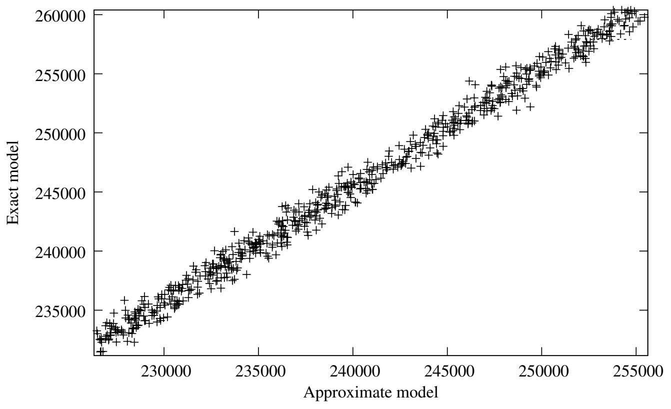

Figure 3. Correlation between the exact and approximate models.

This collection is a subset of the set of the local optima for the approximate instance.

2. For each sequence of $\mathcal{C}$ (sorted in the order of their exact costs), run the local search in the exact model. Finally return the best solution found.

A natural question is to decide when the first step must be stopped. Several strategies can be considered according to the context in which the algorithm must be run. However, according to recent experiments [12], we know that after n descents are run for the approximate model (n is the number of tasks), several very good local optima are likely to be found. Therefore, n is a good estimate for the size of $\mathcal{C}$. Moreover, if a given number (let us say ten) of descents are run without finding new local optimum, we can also stop the first phase, assuming that most local optima have been found. Finally, in a context in which the algorithm has an imperative time limit, phase 1 can be stopped when half time is passed.

The number of descents of the second phase mainly depends on the running time allowed to the algorithm. In a context where the algorithm must very quickly return a good solution and can improve it if more CPU time is eventually allowed, we can imagine a variant of the algorithm that loops between phase 1 and phase 2.

4.2. Experimental tests

The aim of this section is to experimentally validate the assumptions made in the previous subsection and in particular the correlation between the exact and approximate instances. Clearly, this correlation highly depends on the properties of the breaks. We can build instances where the optimization mainly consists of packing the tasks between the breaks. Since, we are interested in periodic breaks, we consider here that the period T is significantly less than the total processing time $P = \sum_i p_i$, which is the case of ILOG instances.

Figure 3 shows the correlation of the costs of random sequences when they are evaluated in the exact or in the approximate model. In the graph, each point is

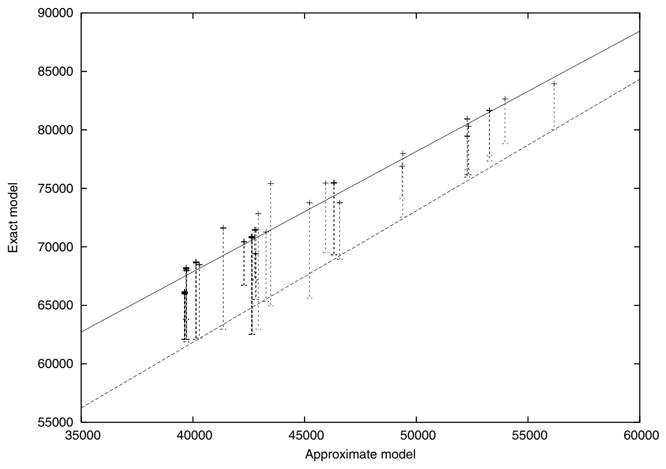

FIGURE 4. Correlation of the local optima.

related to a random sequence: its abscissa is the cost in the approximate model while the ordinate is the cost in the exact model. It clearly shows that the approximation is very good at the scale of the whole solution space.

Of course we are interested in near-optimal solutions, so we have to verify that the correlation is still true for good solutions. Figure 4 illustrates the correlation between the local optima. Each vertical segment represents a local optimum σ of $\mathcal{C}$, the abscissa x_σ is the cost in the approximate model. The ordinate y_σ^+ of the upper endpoint corresponds to the cost of the sequence σ in the exact model. The lower endpoint corresponds to the value y_σ^- of the solution found after the local search descent starting from σ in the exact model. We observe that most local optima for the approximate model are not local optima for the exact model so that the phase 2 local search is indeed necessary: the mean improvement due to phase 2 is 14%. More interestingly, the better local optima for the approximate model lead to the better local optima in the exact model. In order to have a better evaluation of this correlation, we ranked our solutions according to x_σ and y_σ^- and we computed the correlation between the indices of the two rankings. For our instances, the correlation is between 87.1% and 99.9%, which confirms the appearance of Figure 4. This correlation explains why phase 2 of the algorithm starts with the best sequences of the approximate model.

Figure 4 also shows that the deviation between the best and the worst local optima is greater than 30%, which means that if few random local search are met, they might be of bad quality.

Table 2 compares the mean computation time (over 40 runs) of the three possible types of descents: the two procedures called in each phase of our algorithm and the basic descent of Section 3.3 which consists of a local search in the exact model starting with a random sequence. Unsurprisingly, descent in phase 1 is an order of magnitude faster than a descent in the exact model (a descent either from $\sigma \in \mathcal{C}$

Table 2. Mean computation time for each type of descent.

Instance	Number of tasks	Phase 1	Phase 2	From random sequence
STC-BROS-15	30	0.03s	0.15s	0.30s
STC-BROS-15a	30	0.03s	0.17s	0.31s
FS-1	50	0.30s	4.81s	7.09s
FS-2	50	0.33s	4.68s	8.10s
FS-3	50	0.26s	2.14s	4.15s
STC-BROS-41	90	2.03s	30.6s	34.7s
STC-BROS-41a	90	1.89s	26.1s	33.8s

Table 3. Performance comparison.

Instance	Number of tasks	Phase 1 & Phase 2		From random sequence		Best known
STC-BROS-15	30	25696	2.06s	25696	1.20s	25696
STC-BROS-15a	30	6678	2.01s	6678	1.26s	6678
FS-1	50	12202	30.6s	13335	27.9s	12202
FS-2	50	14182	33.2s	14752	32.4s	14182
FS-3	50	107841	18.9s	107883	16.7s	107611
STC-BROS-41	90	71077	179s	73192	137s	70868
STC-BROS-41a	90	26322	156s	27475	133s	26322

or from a random sequence). Moreover, when starting with a sequence $\sigma \in \mathcal{C}$, the descent in the exact model is shorter (in the number of iterations) and thus faster than a descent from a random sequence.

Table 3 compares the two approaches. The two-phase algorithm runs 40 descents in phase 1 and 4 descents in phase 2. It is compared to the average best solution found after 4 descents in the exact model and to the best solution known for the instance. In each case, the two-phase approach finds better solutions, which establishes the usefulness of the first step. Indeed, the first step is useful to have an approximate idea of the solution space, which guides the second step to find the more promising local optima. On the contrary, the approach that starts from random sequences is more myopic so that more descents are on average required to find equivalent solutions.

5. Conclusion

This paper illustrates how existing algorithms for a theoretical scheduling problem can be adapted in order to deal with practical constraints, namely efficiency constraints. These constraints are somehow secondary in comparaison to disjunctive constraints but practitioners usually requires them to be satisfied in an operational context. A theoretical issue is to efficiently manage a large amount of information: with specific data structure, we derive a polynomial algorithm while there is a pseudopolynomial number of efficiency variations.

In a practical approach, we first approximate these moderately tractable constraints in order to have a fast approximate image of the solution space, which is then used to find good solutions in the exact model. This two-step approach is shown to be quite effective.

Interestingly, the lower bounds for the basic problems (*e.g.* [12]) can also be adapted to deal with efficiency constraints. A further work could consists of implementing a branch-and-bound algorithm for this problem. Other natural issues are to extend the approach of this paper to solve more general problems, such like the case where extra breaks may be added to the periodic efficiency curves.

From a theoretical point of view, we can also search for some performance guarantee on the ratio between the local optima of the exact and approximate models.

Acknowledgements.

This study stems from a consulting agreement between the author and ILOG Inc. The author would like to thank the referees for their help to improve the presentation of the paper.

References

[1] N. Brauner, Y. Crama, A. Grigoriev and van de Klundert, A framework for the complexity of high-multiplicity scheduling problems. *J. Combin. Optim.* **9** (2005) 313–323.

[2] J.J. Clifford and M.E. Posner, High multiplicity in earliness-tardiness scheduling. *Oper. Res.* **48** (2000) 788–800.

[3] M.R. Garey, R.E. Tarjan and G.T. Wilfong, One-processor scheduling with symmetric earliness and tardiness penalties. *Math. Oper. Res.* **13** (1988) 330–348.

[4] A. Grosso, F. Della Croce and R. Tadei, An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. *Oper. Res. Lett.* **32** (2004) 68–72.

[5] C. Hanen and A. Munier, *Cyclic scheduling on parallel processors: an overview*, Scheduling Theory and its applications, edited by P. Chrétienne, E.G. Coffman, J.K. Lenstra and Z. Liu, John Wiley & Sons (1995) 193–226.

[6] Y. Hendel and F. Sourd, Efficient neighborhood search for the one-machine ealiness-tardiness scheduling problems. *Eur. J. Oper. Res.* **173** (2006) 108–119.

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

[8] ILOG Inc., *ILOG Scheduler 6.0 User's Manual and Reference Manual* (October 2003).

[9] C.-Y. Lee, *Machine scheduling with availability constraints*, Handbook of scheduling: Algorithms, models and performance analysis, edited by J.Y.-T. Leung, Chapman & Hall/CRC (2004).

[10] W. Nuijten, T. Bousonville, F. Focacci, D. Godard and C. Le Pape, Towards a real-life manufacturing scheduling problem and test bed, in *Proceedings of PMS'04*, `http://www2.ilog.com/masclib` (2004) 162–165.

[11] F. Sourd, Optimal timing of a sequence of tasks with general completion costs. *Eur. J. Oper. Res.* **165** (2005) 82–96.

[12] F. Sourd and S. Kedad-Sidhoum, A new branch-and-bound algorithm for the minimization of earliness and tardiness on a single machine, in *7th workshop on Models and Algorithms for Planning and Scheduling Problems* (2005) 258–261.