Scheduling a sequence of tasks with general completion costs

Francis Sourd

Scheduling a sequence of tasks with general completion costs

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

Abstract

Scheduling a sequence of tasks — in the acceptation of finding the execution times — is not a trivial problem when the optimization criterion is irregular as for instance in earliness-tardiness problems. This paper presents an efficient Dynamic Programming algorithm to solve the problem with general cost functions depending on the end time of the tasks, idle time costs and variable durations also depending in the execution time of the tasks. The algorithm is also valid when the precedence graph is a tree and it can be adapted to determine the possible execution windows for each task not to exceed a maximum fixed cost.

Subject : Production/Scheduling; Sequencing; Deterministic; Single machine; Irregular criterion. Mathematics; Piecewise linear; dynamic programming with cost functions.

1 Introduction

Just-in-Time scheduling has interested both practitioners and researchers for over a decade. A very common idea is to recognize that a job that completes either tardily or early in a schedule incurs extra costs. Therefore, a usual model is to introduce earliness and tardiness penalties per time unit for each task and the objective is to minimize the sum of all the earliness and tardiness costs.

However, such a model may be insufficient. Very often, the earliness and tardiness costs are not linear on the whole time horizon. For example, practitioners sometimes want to model several *good* time periods during which a task would preferably be processed, but with *bad* time periods in between the good periods. Moreover, they also have to deal with idle periods : in schedules minimizing the earliness-tardiness costs, periods of inactivity are generally inserted but in practice, these periods when no work is done have an extra cost that cannot be ignored in the model and must be penalized.

In this paper, the single-machine problem with general completion costs and idle period penalties is studied. More precisely, we will consider the key problem where the tasks are already sequenced. This problem is very important because most scheduling algorithms first rank the tasks by the mean of either a (meta)heuristic or an enumeration scheme and next determine the optimal — if possible — timing for the sequenced tasks. For example, both the branch-and-bound algorithm by Hoogeveen and van de Velde [6] and the tabu

search by Wan and Yen [11] are based upon this approach to solve the single machine problem with earliness/tardiness penalties.

When the completion costs are nondecreasing — criteria such as the flow time and the total tardiness — and when the cost of idle period is nondecreasing with the length of the period, the problem is obvious : each task is scheduled as early as possible when its predecessor in the sequence is completed.

The pure earliness-tardiness case (without idle time penalties) can be formulated as a linear program [4] but this problem can be more efficiently solved in $O(n \log n)$ time by a direct algorithm based on the blocks of adjacent tasks [5, 3, 10]. Chrétienne and Sourd [2] presented a generalization of this algorithm when the cost functions are convex and when the order between the tasks is only partial, that is given by an arbitrary acyclic precedence graph between the tasks. When the minimization criterion is the maximum cost instead of the sum of all the costs, the problem of finding optimum start times for a sequence of tasks can be efficiently solved with general cost functions [8].

Our problem is also related to the project scheduling problem with irregular starting time costs [9]. However, the approach adopted by Mörhing et al. [9] requires to explicitly define the cost of each task at any time point so that the time horizon of the schedule appears in the complexity of the algorithm.

The algorithm presented in this paper is based on dynamic programming. It is faster than the general algorithm of Möhring et al. especially when cost functions are piecewise linear. In such a situation, the algorithm is polynomial in the number of segments of the cost functions given in input of the algorithm. Moreover, our algorithm is able to deal with durations depending on the execution time of the task, which can be very useful to model breaks or transportation activities.

Section 2 presents with more mathematical details the problem studied in the paper. It also considers modelization questions. Section 3 is devoted to the solution of the problem by dynamic programming; the computational complexity is studied when the cost function are piecewise linear. Finally, in Section 4, we adapt the dynamic programming approach to compute the possible start times of all the activities such that a fixed maximum total cost is not exceeded.

2 Problem description

2.1 Problem definition

The problem is to find the execution times of n sequenced tasks denoted by $T_1, T_2, \cdots, T_n$ that is T_i can start only after T_{i-1} is completed. In a feasible schedule, S_i and C_i respectively denote the start time and the end time of T_i. The relationship between S_i and C_i is assumed to be known in advance, $C_i - S_i$ being the duration of T_i. More precisely, it is assumed that S_i is a continuous nondecreasing function of C_i, which is denoted by $S_i = S_i(C_i)$. In other words, the later a task starts, the later it completes and there is only one possible start time for a given end time. Note that the function is not required to be strictly increasing, which is of great importance to deal with breaks. However, for simplicity of the proof, it is required to be continuous but usual non-continuous functions can be seen as the limit of a sequence of continuous functions. We will give an example

in § 2.3.

For each task T_i, a cost function f_i depending on the completion time C_i is given. In order to avoid situation where a task would have to be scheduled at $\pm\infty$, f_i is required to be nonincreasing on some interval $(-\infty, a_i]$ and nondecreasing on some interval $[b_i, \infty)$.

If T_{i+1} does not start just at the completion time of T_i, there is an *idle period* of length $S_{i+1} - C_i$ between the two tasks. The cost of the idle period is measured by the value $w_i(S_{i+1} - C_i)$, where w_i is a function defined on $\mathbb{R}^+$. w_i need not be nondecreasing on $\mathbb{R}^+$ but as for functions f_i, it is required to be nondecreasing on some interval $[c_i, +\infty)$ to avoid to have some tasks scheduled at $\pm\infty$. So the total cost is

$$\sum_{1 \leq i \leq n} f_i(C_i) + \sum_{1 \leq i < n} w_i \left(S_{i+1} \left(C_{i+1} \right) - C_i \right) \tag{1}$$

and the aim of the problem is to minimize this cost subject to the precedence constraint $C_i \leq S_{i+1}$.

For example, in the pure earliness-tardiness case, the cost functions are defined as

$$f_i(C_i) = \max \left(\alpha_i \left(d_i - C_i \right), \beta_i \left(C_i - d_i \right) \right)$$

where α_i and β_i are respectively the earliness and tardiness penalties per unit time.

2.2 Discontinuities in cost functions

In § 2.1, we already gave some conditions so that no task will be scheduled at $\pm\infty$. However, for any closed interval I in which a task T_i must complete, we want to be sure there exists an end time S_i such that $f_i(S_i) = \min_{t \in I} f_i(t)$. A sufficient condition to ensure this property is to assume that the cost functions are continuous. But as we can see in § 2.3, discontinuities in cost functions are interesting — for example to model breaks. So, the following definition introduces a weaker condition on functions to ensure that the minimum is reached at one point.

Definition 1. *A function f is* continuous from the minimum *if, for each point of its definition domain, it is continuous from the left or from the right. Moreover, f must have a finite number of non-continuous points on any bounded interval and, for each t at which f is not continuous, f must satisfy :*

- *if f is not continuous from the left, there is $\delta > 0$ such that for any t' such that $t - \delta < t' < t$ then $f(t') \geq f(t)$.*
- *if f is not continuous from the right, there is $\delta > 0$ such that for any t' such that $t < t' < t + \delta$ then $f(t') \geq f(t)$.*

For example, the ceiling function that returns $\lceil x \rceil$ is continuous from the minimum whereas the flooring functions ($\lfloor x \rfloor$) is not. The main interest of using such functions is shown in the following lemma.

Lemma 1. *Let f be a function that is continuous from the minimum. For any closed and bounded interval I, there exists some $t^\star \in I$ such that $f(t^\star) = \min_{t \in I} f(t)$.*

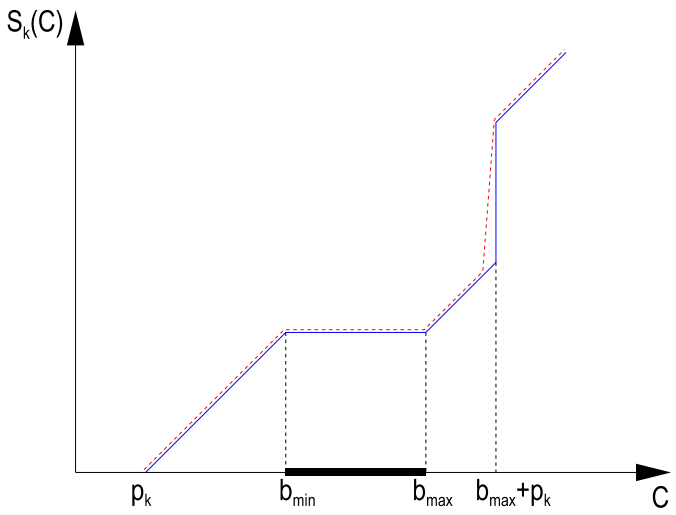

Figure 1: Start function $S_k(C_k)$ in presence of a break $(b_{\min}, b_{\max})$ and its transformation into a continuous function

Proof. Let $t_1, \cdots, t_k$ be the noncontinuous points of f in the interval I. Since f is continuous from the minimum, for each t_i, there is an open interval I_i such that t_i is an end point of I_i and for any $t' \in I_i$, $f(t') \geq f(t_i)$. Let $\hat{I} = I - \bigcup_{i=1}^{k} I_i$. We have that $\min_{t \in I} f(t) = \min_{t \in \hat{I}} f(t)$. Moreover, $\hat{I}$ is a finite union of disjoint closed and bounded intervals and f is continuous on each interval. So there is some $t^\star$ so that $f(t^\star) = \min_{t \in \hat{I}} f(t)$. □

If f and g are both continuous from the minimum, the function $f+g$ is also continuous from the minimum. If h is continuous and is either nondecreasing or nonincreasing, $f \circ h$ is continuous from the minimum. But, with g that is simply continuous from the minimum, $f \circ g$ may not be continuous from the minimum.

2.3 Time windows and breaks

Time window and break constraints often appear in practical models. They can be integrated in our model my setting the costs to ∞ (*i.e.* any upper bound on the solution) on some time intervals when the execution of the task is forbidden. For example, if task T_k is constrained to be entirely executed in the time window $[s_{\min}, e_{\max}]$, its minimum end time $e_{\min}$ is $\min\left(S_k^{-1}\left(\{s_{\min}\}\right)\right)$ so we set the cost function f_k to ∞ on intervals $(-\infty, s_{\min})$ and $(s_{\max}, +\infty)$. If f_k is continuous from the minimum, the modified function is still continuous from the minimum. In the same way, the task can be constrained to have several possible time windows (while being again constrained to be entirely executed in only one time windows).

A break is a time interval $(b_{\min}, b_{\max})$ during which the execution of a task can be suspended at the break start $b_{\min}$ and restarted at the break end $b_{\max}$. For a task T_k

whose processing time is p_k, the start function S_k is (see Figure 1) :

$$S_k(C) = \begin{cases} C - p_k & \text{if } C < b_{\min} \\ b_{\min} - p_k & \text{if } b_{\min} \leq C < b_{\max} \\ C - p_k - (b_{\max} - b_{\min}) & \text{if } b_{\max} \leq C < b_{\max} + p_k \\ C - p_k & \text{if } b_{\max} + p_k \leq C \end{cases}$$

Moreover, since T_k cannot complete during the break, its cost function f_k must be set to ∞ on the interval $(b_{\min}, b_{\max})$. However, we note that the above-defined function S_k is noncontinuous at $C = b_{\max}+p_k$ while our mathematical model requires it to be continuous. But we can remark that if T_k ends at $b_{\max} + p_k - \epsilon$ (for some small $\epsilon > 0$), the fraction ϵ/p_k of T_k is processed just before the break, which is not desirable in practise. We can prevent such a situation by fixing the minimum fraction α_k of T_k that must be processed before the task may be interrupted by the break. T_k can be prevented from completing in $(b_{\max} + (1 - \alpha_k)p_k, b_{\max} + p_k]$ by setting f_k infinite on this interval and S_k can be transformed into a continuous function by modifying it only on this interval, as shown by Figure 1.

2.4 Constant idle time cost

When the idle time costs are linear and task-independent, that is there is some α such that for any $i \in \{1, \cdots, n\}$, $w_i(t) = \alpha t$, and when the durations of the tasks are time-independent, that is for any i there is a constant p_i such that $S_i = C_i - p_i$, the problem can be reduced to a problem without idle time cost. Indeed, the total cost given by Equation (1) is now

$$\sum_{1 \leq i \leq n} f_i(C_i) + \alpha \sum_{1 \leq i < n} C_{i+1} - p_{i+1} - C_i$$

that is

$$\sum_{1 \leq i \leq n} f_i(C_i) + \alpha C_n - \alpha C_1 - \alpha \sum_{1 < i \leq n} p_i$$

Since the last term is a constant, the minimization problem is equivalent to the problem with null idle time costs in which the costs functions are :

$$\hat{f}_i(C) = \begin{cases} f_1(C) - \alpha C & \text{for } i = 1 \\ f_i(C) & \text{for } 1 < i < n \\ f_n(C) + \alpha C & \text{for } i = n \end{cases}$$

We are going to see in next section that when there is no idle time the recurrence equation of the Dynamic Program is much simpler.

3 Solving the problem by dynamic programming

We are going to show that this problem can be solved by dynamic programming. For any $k \in \{1, \cdots, n\}$ and any $t \in \mathbb{R}$, $P_k(t)$ denotes the subproblem in which :

- the sequence of tasks is the subsequence $T_1, \cdots, T_k$ and

- we add the additional constraint that T_k completes at t, that is $C_k = t$.

$\Sigma_k(t)$ is the minimum cost of the solutions of $P_k(t)$. Clearly, the optimal cost of the whole problem is $\min_{t\in\mathbb{R}} \Sigma_n(t)$.

3.1 Recursive relationship

We obviously have that $\Sigma_1(t) = f_1(t)$. For any $k \geq 1$, $\Sigma_{k+1}(t)$ can be expressed in function of Σ_k. Indeed, if T_{k+1} is assumed to complete at time t then T_k must complete at some time t' that is less than or equal to the start time of T_{k+1} that is $S_{k+1}(t)$. The minimum cost of the schedule such that $C_k = t'$ and $C_{k+1} = t$ is $\Sigma_k(t') + w_k\left(S_{k+1}(t) - t'\right) + f_{k+1}(t)$. $\Sigma_{k+1}(t)$ is then given by minimizing upon all the possible values for t':

$$\Sigma_{k+1}(t) = \min_{t' \leq S_{k+1}(t)} \left(\Sigma_k(t') + w_k\left(S_{k+1}(t) - t'\right)\right) + f_{k+1}(t) \tag{2}$$

Theorem 2. *For any integer $k \in \{1, \cdots, n\}$, the function Σ_k is well defined and is continuous from the minimum.*

Proof. We here show that for any function f that is continuous from the minimum and nonincreasing on some interval $(-\infty, a_f)$, the function $g(t) = \min_{t'\leq t} f(t')$ is well defined, is continuous from the minimum and is nonincreasing on some interval $(-\infty, a_g)$ — this last point is obvious. With this result, since the sum of two functions continuous from the minimum is continuous from the minimum and since the composition by a continuous nondecreasing function preserves the continuity from the minimum, the theorem is easily proved by induction.
So, let us consider the function $g(t) = \min_{t'\leq t} f(t')$. Since f is nonincreasing on the interval $(-\infty, a_f)$, for any $t < a_f$, $g(t) = f(t)$. For any $t \geq a_f$, $[a_f, t]$ is a closed and bounded interval of $\mathbb{R}$ so that there is some $t^\star \leq t$ such that $g(t) = f(t^\star) = \min_{a_f \leq t' \leq t} f(t') = \min_{t'\leq t} f(t')$. That shows that $g(t)$ is well defined for any $t \in \mathbb{R}$. Let us now define $E = \{t \,|\, g(t) = f(t)\}$. E is a closed set. Clearly, g is continuous from the minimum at any point of the interior of E and at any point in $\mathbb{R} - E$. What is left to prove is that g is continuous from the minimum at any point of the boundary between E and $\mathbb{R} - E$. Let x be such a point (x is in the closed set E). If f is not continuous from the right at x, this means that f is strictly greater than $f(x)$ on some interval $(x, x + \delta_1)$ and g is constant and for any $t \in (x, x + \delta_1)$, $g(t) = f(x) = g(x)$. If f is continuous from the right at x, for any t in some interval $(x, x + \delta_2)$, $g(t) = \min_{x\leq t'\leq t} f(t')$. So in both cases, g is continuous from the right at x. As g is nonincreasing, g is continuous from the minimum at x. □

From the assumption on the cost functions f_i and w_i, the minimum of Σ_n is in the interval $[\min_i a_i - \max_i c_i, \max_i b_i + \max_i c_i]$. So, Theorem 2 and Lemma 1 prove the existence of some time point C_n such that $\Sigma_n(C_n) = \min_{t\in\mathbb{R}} \Sigma_n(t)$. This time C_n is the completion time of T_n. To compute C_{n-1}, we must add the constraint that $C_{n-1} \leq S_n = S_n(C_n)$: C_{n-1} is a value minimizing Σ_{n-1} on $(-\infty, S_n]$. $C_{n-2}, \cdots, C_1$ are computed by iterating this process.

Finally, we can remark that this dynamic programming approach is also valid when the order between the tasks in not given by a complete chain but by an "intree" precedence

graph. In this problem, each task must have at most one successor (a task without successor is a root of the intree) and several tasks can eventually be processed at the same time, that is there is no resource constraint (Project Scheduling Problem). The recurrence equation is very similar. Here, $P_k(t)$ denotes the subproblem in which the tasks are all the predecessors of T_k that are all the tasks in the subtree of root T_k. We still have the additional constraint that T_k completes at t, that is $C_k = t$. Let Π_k be the set of the direct predecessors of T_k. The recurrence equation is then :

$$\Sigma_k(t) = \begin{cases} f_k(t) & \text{if } \Pi_k = \emptyset \\ f_k(t) + \sum_{i \in \Pi_k} \min_{t' \leq S_k(t)} \left(\Sigma_i(t') + w_{ik}\left(S_k(t) - t'\right)\right) & \text{otherwise} \end{cases}$$

where w_{ij} is the idle time cost between the end of T_i and the start of T_j. To run the dynamic program associated with this equation, the tasks are to be numbered in a topological order compatible with the precedence graph.

For an outtree precedence graph — each task has at most one predecessor — we can use the above-described algorithm by reversing the time scale.

3.2 Piecewise linear functions

The computability and complexity of this algorithm strongly depend on the cost functions given in input. Hereafter, we will study the — somewhat general — case where all the functions are piecewise linear. Each piecewise linear function is supposed to be given as an ordered segment list. For each segment in the list, five values are given corresponding to its definition interval, its slope and the coordinates of one point in the segment. We of course assume that the definition intervals of the segments do not overlap. So the number of segments is a good indicator of the amount of information given in input to describe a piecewise linear function. In this paper, the number of segments of a piecewise linear function f is denoted by $\|f\|$.

For the sake of simpler notations, if a piecewise linear function f is not defined on $\mathbb{R}$, we are going to say that for any real value t outside the definition domain, $f(t) = \infty$. In other words, we transform the initial function into a piecewise linear function defined on $\mathbb{R}$ with constant segments equal to ∞. Clearly, if f has n segments, the transformed function has $O(n)$ segments. Moreover, the transformed function has the property that the rightmost point of a segment — if not infinite — has the same abscissa than the leftmost point of the next segment. Such a point will be hereafter called a *breakpoint* of the function. As Theorem 2 ensures that the computed functions are still continuous from the minimum, we can avoid to wonder if the endpoints of the segments are open or closed : for any discontinuity, the discontinuity point belongs to the segment such that the function is continuous from the minimum.

We now present a series of lemmas about operations involving piecewise linear functions and their computation. In these lemmas, f_1 and f_2 denote two arbitrary piecewise linear functions with respectively $n_1 = \|f_1\|$ and $n_2 = \|f_2\|$ segments. To avoid any possible confusion, note that in the text of the following lemmas, f_1 and f_2 are not the cost functions of tasks T_1 and T_2 defined in Section 2.

Lemma 3. *The function $f(t) = f_1(t) + f_2(t)$ is a piecewise linear function with $O(n_1 + n_2)$ segments. It can be computed in $O(n_1 + n_2)$ time.* □

Lemma 4. *The function $f(t) = \min(f_1(t), f_2(t))$ is a piecewise linear function with $O(n_1+n_2)$ segments. It can be computed in $O(n_1 + n_2)$ time.*

Proof. Clearly, t is a breakpoint for f only if t is a breakpoint for f_1 or f_2 or if $f_1(t) = f_2(t)$ (and $f_1'(t) \neq f_2'(t)$). f_1 and f_2 have respectively $O(n_1)$ and $O(n_2)$ breakpoints. So there are $O(n_1 + n_2)$ breakpoints for f that are due to breakpoints for f_1 and f_2. Between two such breakpoints, both f_1 and f_2 are linear so either these two functions are identical (and there is no breakpoint) or there is zero or one intersection point so that f has at most one breakpoint on this open interval. In conclusion, f has $O(n_1 + n_2)$ breakpoints. It can obviously be computed in $O(n_1 + n_2)$. □

Lemma 5. *If f_2 is continuous and nondecreasing, the function $f(t) = f_1(f_2(t))$ is a piecewise linear function with $O(n_1 + n_2)$ segments. It can be computed in $O(n_1 + n_2)$ time.*

Proof. Since f_2 is continuous and nondecreasing, for each breakpoint t of f_1, the set of real values $S_t = \{t' \in \mathbb{R} \mid f_2(t') = t\}$ is either a single point or a closed interval depending on the slope of f_2 at t'. Let us consider the sorted list L that contains the breakpoints of f_2 and for each breakpoint t of f_1 the endpoints of S_t (that are one or two points). This list contains $O(n_1 + n_2)$ points. In the open interval between two consecutive points of L, the function f is clearly linear as the composition of two linear functions. So f is piecewise linear with $O(n_1 + n_2)$ segments. The following algorithm, very similar to the algorithm to merge sorted arrays, compute the sorted list in $O(n_1 + n_2)$ time :

```
let t1 be the first breakpoint of f1
let t2 be the first breakpoint of f2
let L ← ∅
while t1 or t2 exists do
    if f2(t2) ≤ t1 then
        add t2 to L
        let t2 be the next breakpoint of f2
        if f2(t2) = t1 then
            let t1 be the next breakpoint of f1
    else
        let t such that f2(t) = t1
        add t to L
        let t1 be the next breakpoint of f1
```

So the function f can be computed in $O(n_1 + n_2)$ time. □

The following operation between f_1 and f_2 can be viewed as the composition between f_1 and f_2^{-1}. It will be used in Section 4.

Lemma 6. *If f_2 is continuous and nondecreasing, the function $f(t) = \min_{t=f_2(t')} f_1(t')$ is a piecewise linear function with $O(n_1 + n_2)$ segments. It can be computed in $O(n_1 + n_2)$ time.*

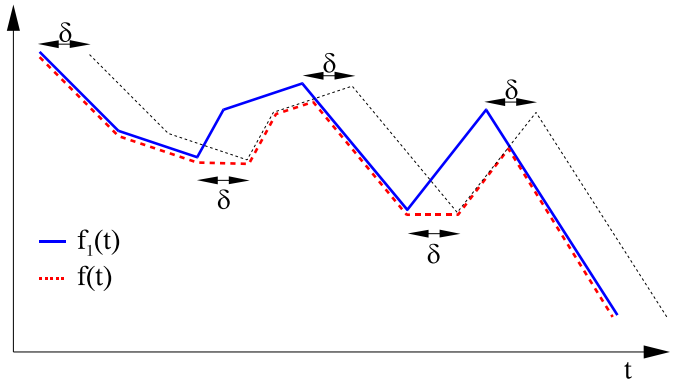

Figure 2: $f(t) = \min_{t-\delta \leq t' \leq t} f_1(t')$

Proof. The definition domain of f_2, that is *a priori* $\mathbb{R}$, can be divided into a partition $I^{\text{inc}} \cup I^{\text{step}}$ such that f_2 is strictly increasing on I^{inc} and is stepwise on I^{step}. With f_2^{inc} (resp. f_2^{step}) being the function f_2 with the definition domain restricted to I^{inc} (resp. I^{step}), we have that

$$f(t) = \min \left(\min_{t=f_2^{\text{inc}}(t')} f_1(t'), \min_{t=f_2^{\text{step}}(t')} f_1(t') \right) \tag{3}$$

The first term is equal to $\min(f_1(f_2^{\text{inc}^{-1}}(t))$. From Lemma 5, it has $O(n_1 + n_2)$ segments and can be computed in $O(n_1 + n_2)$ time. We can compute in $O(n_1)$ time, all the $\min_{t=f_2^{\text{step}}(t')} f_1(t')$ corresponding to each interval of the definition domain of f_2^{step}. These values eventually modify the value of the piecewise linear function given by the first term of (3) at some of its irregular points. There is at most n_2 such modifications. □

Lemma 7. *Let δ be a non-negative real value. The function $f(t) = \min_{t-\delta \leq t' \leq t} f_1(t')$ is a piecewise linear function with at most n_1 segments. It can be computed in $O(n_1)$ time.*

Proof. The construction of f is illustrated by Figure 2. For each breakpoint t_i $(1 \leq i < n_1)$ of the function f_1, let h_i be the constant function defined on the interval $[t_i, t_i + \delta)$ equal to the value $\lim_{\theta \to t_i^-} f_1(\theta)$. For any real value t, the global minimum of the function f_1 on the interval $[t - \delta, t]$ is reached either at t or at $t - \delta$ or at a local minimum of f_1 in the interval. Such a local minimum can only be a breakpoint of f_1 so that we have

$$f(t) = \min\left(f_1(t), f_1(t-\delta), h_1(t), h_2(t), \cdots, h_{n_1-1}(t)\right)$$

Let $h(t)$ be the stepwise function defined by $\min_{1 \leq i < n_1} h_i(t)$. Since the breakpoints of f_1 are given ordered in input and since the definition intervals of all h_i functions have the same length (δ), h can be computed in $O(n_1)$ time and has $O(n_1)$ steps. Therefore, from Lemma 4, the piecewise linear function defined by $f(t) = \min\left(f_1(t), f_1(t-\delta), h(t)\right)$ can be computed in $O(n_1)$ time and has $O(n_1)$ segments. □

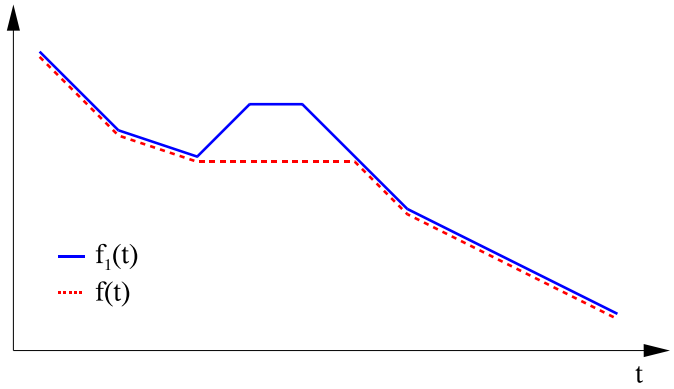

Figure 3: $f(t) = \min_{t' \leq t} f_1(t')$

When δ is greater than $t_{n_1} - t_1$, we clearly have that $f(t) = \min_{t' \leq t} f_1(t')$ which gives the following corollary illustrated by Figure 3 :

Corollary 8. *The function $f(t) = \min_{t' \leq t} f_1(t')$ is a piecewise linear function with at most $O(n_1)$ segments. It can be computed in $O(n_1)$ time.* □

We can even show that the function f has at most n_1 segments. The following lemma is similar to Corollaty 8 but with two functions.

Lemma 9. *The function $f(t) = \min_{t_1+t_2 \geq t} f_1(t_1) + f_2(t_2)$ is a piecewise linear function with $O(n_1 n_2)$ segments. It can be computed in $O(n_1 n_2)$ time.*

Proof. We are going to prove that $\hat{f}(t) = \min_{t_1+t_2=t} f_1(t_1) + f_2(t_2)$ is a piecewise linear function with $O(n_1 n_2)$ segments that can be computed in $O(n_1 n_2)$ time. Indeed, we clearly have that $f(t) = \min_{t' \geq t} \hat{f}(t')$ and so f can be derived from $\hat{f}$ thanks to Corollary 8 by reversing the abscyssa axis.
Let $I_1, I_2, \cdots, I_{n_1}$ be the definition intervals of each segment of the piecewise linear function f_1. On each interval I_k, f_1 is linear so there are α_k and β_k such that for all $t \in I_k$, $f_1(t) = \alpha_k t + \beta_k$. We can then write $\hat{f}(t) = \min_{1 \leq k \leq n_1} g_k(t)$ with $g_k(t) = \min_{t_1 \in I_k} \alpha_k t_1 + \beta_k + f_2(t - t_1)$. So $g_k(t) = \alpha_k t + \beta_k + \min_{t_1 \in I_k} f_2(t - t_1) - \alpha_k(t - t_1)$. From Lemmas 5 and 7, the last term of function g_k has $O(n_2)$ segments and can be computed in $O(n_2)$ time. So each function g_k has $O(n_2)$ segments and can be computed in $O(n_2)$ time, which shows that f has $O(n_1 n_2)$ segments and can be computed in $O(n_1 n_2)$ time. □

We can give a simple example in which the function $\hat{f}(t)$ defined in Lemma 9 has $\Theta(n_1 n_2)$ segments. f_1 is defined on the interval $[0, n_1 n_2)$ and $f_1(t) = (n_1 + 1)\lceil t/n_1 \rceil$. f_2 is defined on the interval $[0, n_1)$ and $f_2(t) = \lceil t \rceil$. f_1 and f_2 are continuous from the minimum. Since they are both nondecreasing, we have $\hat{f}(t) = \min_{t_1+t_2=t} f_1(t_1) + f_2(t_2)$. As $t_2 \in [0, n_1)$, we have that $t_1 \in (t - n_1, t]$. Let $\alpha = \lfloor t/n_1 \rfloor$ and $\beta = t - \alpha n_1$ that is a real value in $[0, n_1)$. $f_1(\alpha n_1) + f_2(\beta) = \alpha(n_1 + 1) + \lceil \beta \rceil$. If $t_1 > \alpha n_1$ then $f_1(t_1) \geq (\alpha + 1)(n_1 + 1) \geq f_1(\alpha n_1) + f_2(\beta)$. If $t - n_1 < t_1 \leq \alpha n_1$, $f_1(t_1) = \alpha(n_1 + 1)$ and, since f_2 is non decreasing, $f_2(t - t_1) \geq f_2(\beta)$. So the minimum is reached for $t_1 = \alpha n_1$ and $t_2 = \beta$

which means that $f(t) = f_1(\lfloor t/n_1 \rfloor) + f_2(t - \lfloor t/n_1 \rfloor)$ and this function has $n_1 n_2$ steps of length 1 in $[0, n_1 n_2)$.

3.3 Complexity analysis

Now, f_i is again the cost function of task T_i. Using the lemmas of the previous subsection, a simple induction on (2) shows that the time and space complexities of the dynamic program when all the function given in input are piecewise linear are :

$$O\left(n\left(\prod_{i=1}^{n-1}\|w_i\|\right)\left(\sum_{i=1}^{n}\|f_i\| + \sum_{i=2}^{n}\|S_i\|\right)\right) \tag{4}$$

That shows that the algorithm may not be efficient when the idle time cost are not linear.

When the idle time costs are linear, the complexity is $O\left(n\left(\sum_{i=1}^{n}\|f_i\| + \sum_{i=2}^{n}\|S_i\|\right)\right)$. In the pure earliness-tardiness problem, each cost function f_i has two segments so that the complexity of the algorithm is $O(n^2)$. We recall this problem can be solved in $O(n \log n)$.

When the idle time between two consecutive tasks is constrained to be in a given interval, and the cost is linear inside the interval, we can show with Lemma 7 that the complexity is again $O\left(n\left(\sum_{i=1}^{n}\|f_i\| + \sum_{i=2}^{n}\|S_i\|\right)\right)$. This tractable case seems large enough to model a great deal of practicle problems.

4 Optimal filtering algorithm

Filtering algorithms are of key importance in Constraint Programming because their role is to remove from the domain of the variables, the values that cannot lead to a *feasible* solution. Sometimes, the removal of values than cannot lead to an *optimal* solution can be implemented. In constraint-based scheduling [1], a pair of variables is usually devoted to the start and end times of each activity. In this section, we give an algorithm that determines the possible end times for each activity so that the total cost of the schedule is not greater than a given maximum cost. The possible start time can then be directly determined by the functions S_k. This filtering algorithm is said to be optimal because any possible end time rendered by the algorithm corresponds to at least one feasible schedule with a cost not greater than the given maximum cost. In other words, an optimal filtering algorithm keeps all the possible values but only the possible values.

4.1 Problem description

We keep the notations introduced in Section 2 and we add the notation F that represents the maximum possible cost. Therefore, the objective criterion given by (1) is replaced by the following hard constraint :

$$\sum_{1\leq i\leq n} f_i(C_i) + \sum_{1\leq i<n} w_i\left(S_{i+1}\left(C_{i+1}\right) - C_i\right) \leq F \tag{5}$$

The problem is to compute, for each task T_k $(1 \leq k \leq n)$, the set $\mathcal{C}_k(F)$ of all the possible completion times t such that there exists at least one feasible schedule satisfying

the precedence constraints given in Section 2, the cost constraint (5) and the constraint $C_k = t$.

4.2 Algorithm and properties

The function $\Sigma_k(t)$ defined by relationship (2) gives the minimal cost to schedule the *initial* subsequence $T_1, \cdots, T_k$ such that $C_k = t$. Symmetrically, we define $\bar{\Sigma}_k(t)$ as the minimal cost to schedule the *terminal* subsequence $T_k, \cdots, T_n$ such that T_k ends at t, that is $C_k = t$. We set $\bar{\Sigma}_n(t) = f_n(t)$ for any t. If T_k (with $k < n$) ends at t and T_{k+1} ends at t' with $S_{k+1}(t') \geq t$, the minimum cost to schedule $T_k, \cdots, T_n$ is $f_k(t) + w_{k+1}(S_{k+1}(t') - t) + \bar{\Sigma}_{k+1}(t')$. So $\bar{\Sigma}_k(t) = f_k(t) + \min_{S_{k+1}(t') \geq t}(w_{k+1}(S_{k+1}(t') - t) + \bar{\Sigma}_{k+1}(t'))$.

For each task T_k, we can then define the function $\Sigma_k^\star(t)$ that is equal to the minimum cost to schedule all the tasks $T_1, \cdots, T_n$ such that $C_k = t$. Clearly, we have:

$$\Sigma_k^\star(t) = \Sigma_k(t) + \bar{\Sigma}_k(t) - f_k(t) \tag{6}$$

So, $\mathcal{C}_k(F) = \{t | \, \Sigma_k^\star(t) \leq F\} = {\Sigma_k^\star}^{-1}((-\infty, F])$. As for Σ_k, the function $\bar{\Sigma}_k$ is continuous from the minimum. $\Sigma_k^\star$ is also continuous from the minimum since $\bar{\Sigma}_k(t) - f_k(t)$ is continuous from the minimum.

Theorem 10. *$\mathcal{C}_k(F)$ is a closed set.*

Proof. If $\mathcal{C}_k(F)$ is not empty, let us consider an infinite sequence (t_i) such that for each $i \in \mathbb{N}$, $t_i \in \mathcal{C}_k(F)$ and $\lim_{i \to \infty} t_i = t$. Since $\Sigma_k^\star$ is either continuous from the right or from the left, we can extract from this sequence an infine sequence $(\hat{t}_i)$ such that $\left(\Sigma_k^\star(\hat{t}_i)\right)$ has a limit ℓ. Since for each i, $\Sigma_k^\star(\hat{t}_i) \leq F$, we have that $\ell \leq F$. Since $\Sigma_k^\star$ is continuous from the minimum, $\Sigma_k^\star(t) \leq \ell$. Therefore, $\Sigma_k^\star(t) \leq F$ and $t \in \mathcal{C}_k(F)$. □

The following theorem is useful to speed up the computation of $\mathcal{C}_k(F)$ in the earliness-tardiness case without idle time costs — or with a constant idle time cost, which is equivalent (see § 2.4).

Theorem 11. *When all the cost functions are convex and when there is no idle time costs, $\mathcal{C}_k(F)$ is an interval.*

Proof. We first remark that if a function g is convex, the function $f(t) = \min_{t' \leq t} g(t')$ is also a convex function because f and g are equal before the minimum of g and f is constant after the minimum of g (see Figure 4). Therefore, Σ_k is convex as a sum of convex function. So the functions $\bar{\Sigma}_k - f_k$ and $\Sigma_k^\star$ are also convex. The convexity of $\Sigma_k^\star$ proves that $\mathcal{C}_k(F)$ is an interval. □

The study of the computation of $\mathcal{C}_k(F)$ is very similar to what was done in the previous section. However, we must rewrite the definition of $\bar{\Sigma}_k$ as

$$\bar{\Sigma}_k(t) = f_k(t) + \min_{t'' \geq t} \left(w_{k+1}(t'' - t) + \min_{t'' = S_{k+1}(t')} \bar{\Sigma}_{k+1}(t') \right)$$

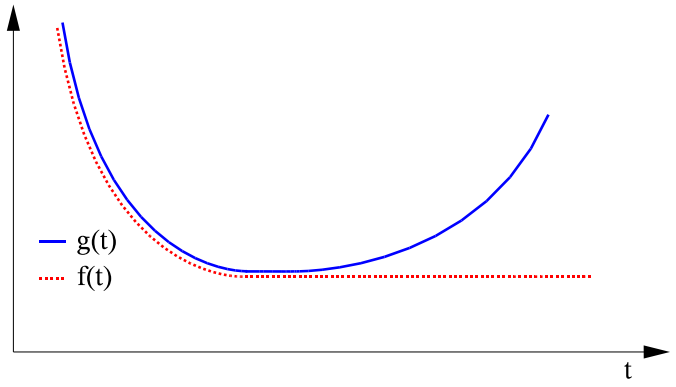

Figure 4: $f(t) = \min_{t' \leq t} g(t')$ is convex if g is convex.

to show with Lemma 6 that the function can be computed with the same time and space complexity than Σ_{k+1} in § 3.3. Therefore, $\Sigma_k^\star$ can also be computed with the same complexity. Since the time to compute $\mathcal{C}_k(F)$ is proportional of the number of segments of $\bar{\Sigma}_k$, the total time to compute the sets $\mathcal{C}_k(F)$ for all the tasks $T_1, \cdots, T_n$ is

$$O\left(n\left(\prod_{i=1}^{n-1}\|w_i\|\right)\left(\sum_{i=1}^{n}\|f_i\| + \sum_{i=2}^{n}\|S_i\|\right)\right)$$

which is the same complexity as the one given by (4) to compute the only optimal value of the optimization problem. We of course have the same simplification in the formula when the idle time costs are linear.

5 Conclusion

We presented a Dynamic Programming algorithm to schedule — *ie* to time — a sequence of tasks in order to minimize the total cost, including idle time costs. The algorithm is still valid for the project scheduling problem without resource constraint and with an intree or outtree precedence graph. An interesting point of this algorithm is that it can be very easily implemented. For example, the problem with piecewise linear cost functions was implemented in a few lines of code using the piecewise linear function facilities of ILOG Scheduler 5.2 [7]. Moreover, it seems to be very efficient for large classes of practical instances.

This algorithm can be adapted — with no extra computational cost — to get information on the possible execution time window for each task so that a maximum fixed cost is not exceeded.

Both information on the minimum cost of a sequence and on the possible time windows should be very useful to determine an optimal sequence by a branch-and-bound algorithm. Further research will focus on such an algorithm and its use in solving shop problems.

Acknowledgements

Part of the work was done while the author was working for ILOG.

References

[1] Ph. Baptiste, C. Le Pape, and W. Nuijten, *Constraint-based scheduling : Applying constraint programming to scheduling problems*, International Series in Operations Research and Management Science, vol. 39, Kluwer Academic Publishers, Boston, 2001.

[2] Ph. Chrétienne and F. Sourd, *Scheduling with convex cost functions*, Theoretical Computer Science (2002), to appear.

[3] J.S. Davis and J.J. Kanet, *Single machine scheduling with early and tardy completion costs*, Naval Research Logistics **40** (1993), 85–101.

[4] T.D. Fry, R.D. Armstrong, and J.H. Blackstone, *Minimize weighted absolute deviation in single machine scheduling*, IIE Transactions **19** (1987), 445–450.

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

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

[7] ILOG, S.A., *Ilog Scheduler 5.2 user's manual and reference manual*, December 2001.

[8] S. Lakshminarayan, R. Lakshmanan, R.L. Papineau, and R. Rochete, *Optimal single-machine scheduling with earliness and tardiness penalties*, Operations Research **26** (1978), no. 6, 1079–82.

[9] R.H. Möhring, A.S. Schulz, F. Stork, and M. Uetz, *On project scheduling with irregular starting time costs*, Operations Research Letters **28** (2001), 149–154.

[10] W. Szwarc and S.K. Mukhopadhyay, *Optimal timing schedules in earliness-tardiness single machine sequencing*, Naval Research Logistics **42** (1995), 1109–1114.

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