Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Estatística

An Algorithm for Insertion of Idle Time in the Single-Machine Scheduling Problem with Convex Cost Functions

Emerson C. Colin
Roberto C. Quinino

Relatório Técnico
RTP- 01/2003
Série Pesquisa

An Algorithm for Insertion of Idle Time in the Single-Machine Scheduling Problem with Convex Cost Functions

Emerson C. Colin

Departamento de Engenharia de Produção, Universidade de São Paulo

Av. Prof. Almeida Prado, 531, 2o. andar, CEP 05508-900, São Paulo, SP, Brazil[*]

Roberto C. Quinino

Departamento de Estatística, Universidade Federal de Minas Gerais

Av. Antônio Carlos, s/n, ICEX, CEP 31270-901, Belo Horizonte, MG, Brazil

Abstract

This paper addresses the problem of optimally inserting idle time into a single-machine schedule when the sequence is fixed and the cost of each job is a convex function of its completion time. We propose a pseudo-polynomial time algorithm to find a solution within some tolerance of optimality in the solution space. The proposed algorithm generalises several previous works related to the subject.

Keywords: Scheduling; Single-machine scheduling; Idle time; Convex cost; Fixed sequence.

1. Introduction

The greatest part of the models in scheduling considers linear objective functions, hereafter objectives. On the other side, many times, a model considering

nonlinearities in the objective can improve the quality of the representation of the real problem without increasing too much the complexity of its solution.

We can mention many industrial settings where nonlinearities can significantly improve the representation of the problem. A first example can be cited in the delivery of perishables such as fruits, bakery, or dairy products to the retailer where earliness costs grow more than proportionally to the earliness. Another example relates to the goodwill costs associated to the delivery of medicines to pharmacies and wholesalers. Safety inventories generally absorb a certain level of tardiness, but from a specific point on, the costs incurred to the patients may be huge.

In this paper we adapt an existing algorithm that treats linear objectives for the nonlinear case, generalising previous results. In a recent survey [9] Davis and Sridharan suggest the treatment given here as an open avenue for further development of algorithms of insertion of idle time. The work is based on [8] that proposes an Algorithm for Insertion of Idle Time (AIIT) in the earliness-tardiness problem. The algorithm presented here follows the same structure of Garey *et al.*'s paper and many definitions are given here for reference.

Let $J = \{1,2,...\}$ be a finite ordered set of indices that represent jobs. Every job $j(j \in J)$ has a specified processing time p_j, a due date d_j, and a convex function $g_j(C_j)$ that defines the cost incurred in the completion time (C_j) of the job.

The work accomplished considers that a sequence has been defined previously by some other appropriate procedure to that end. The scheduling here defines the assignment of the starting time e_j for each job in such way that two jobs cannot be processed at the same time, unless (for the sake of simplicity) at the endpoints of the processing intervals where an overlap can exist. For a particular job *j*, there is a

processing interval defined as $[e_j, e_j + p_j]$ where [···] represents a closed interval. Let W_j be the idle time before starting *j*. The schedule π of a set of jobs *J* is the assignment of the starting times of all jobs, that is, $\pi = \{e_1, e_2, ..., e_{|J|}\}$, where $e_j = C_{j-1} + W_j$ and $C_0 = 0$. A partial schedule π_j is a schedule that contains the *j* first jobs of *J*.

The schedule is accomplished by trying to finish the jobs at their due dates d_j, taking into account that every job requires a certain amount of time for its processing. Idle time W_j can be used between the processing of two consecutive jobs. The objective studied in this paper can be defined as $\min g(\pi) = \sum_{j \in J} g_j(C_j)$. Although the major part of scheduling and sequencing research does not consider nonlinear costs, there are a bunch of papers where nonlinear costs are a matter of concern. Among them we can mention [1], [2], and [4].

The remaining paper is organised as follows: Section 2 describes the literature related to this work. In Section 3 we describe the proposed algorithm and the rationale underlying it. Section 4 gives some details of the computational questions related to the algorithm. Some interesting remarks about the object of this work are discussed in Section 5.

2. Literature Review

The insertion of idle time makes sense only when an earlier completion of the job can decrease the total cost of the schedule. The literature related to the insertion of idle time is always associated with problems containing earliness and tardiness.

The tardiness of a job j is defined as $T_j = \max\{0, C_j - d_j\}$ and the earliness as $E_j = \max\{0, d_j - C_j\}$. For the linear case, w_j (h_j) is the tardiness (earliness) cost.

Probably the first widely published work that considers an AIIT is due to [7]. In a quite clear way, the authors model the problem of insertion of idle time in the asymmetric earliness-tardiness problem $\sum_{j \in J}(h_j E_j + w_j T_j)$ as a linear programming problem. The authors propose a solution procedure that runs in $O(|J|^2)$ time. Some years later several papers where published on the same subject offering different algorithms and different views of the problem [9].

In a distinct paper, [8] propose an AIIT for the symmetric earliness-tardiness problem $\sum_{j \in J}\{w_j(E_j + T_j)\}$ with running time $O(|J|\log|J|)$. In the same paper the sequencing problem associated has been proved to be NP-hard. Colin and Shimizu [5] adapt the Garey-Tarjan-Wilfong's algorithm for the asymmetric case maintaining the running time in $O(|J|\log|J|)$.

Although it had not been clearly evident to us, [9] mention that the timetabler procedure [6] solves the problem $g(\pi) = \sum_{j \in J} g_j(C_j)$ in $O(|J|H)$, where H is the number of units of time in the planning horizon. Here we propose an algorithm for the same problem with a slightly improved running time $O(|J|\max\{|J|, \log(d_{\max}/z)\})$ where z is a tolerance of optimality in the solution space.

It is important to note that idle time insertion algorithms are used along sequencing procedures and computational efficiency is important because it allows the use of the algorithm in every sequence created/tested. All papers mentioned in

this section suppose that the algorithms are used after a fixed sequence has been defined. For a more comprehensive review of the related literature see [9].

3. Algorithm for Insertion of Idle Time

In this section we suggest an AIIT considering a broad class of objective functions. This class can be synthesised as anyone whose costs are represented by a sum of independent convex functions of completion times of the jobs. In fact, the problem about considering a more generic class of function such as pseudoconvex or quasi-convex is the lack of the quasi-convexity property of the sum of these functions. Functions that are not guaranteed to have quasi-convexity properties claim global optimisation procedures that generally are very difficult to solve.

Although idleness is highlighted in the name of the algorithm, its use does not define idle time directly. The algorithm defines starting times; idleness, as previously mentioned, can be found by the following equation: $W_j = e_j - (e_{j-1} + p_{j-1})$.

Let $C_j^* = \arg\min g_j(C_j)$ be the target completion time, i.e. the completion time that minimise the costs related to j. C_j^* will depend on the function that defines the costs associated to j.

Let $B = \{1,2,...,l\}$ be the set of indices of the blocks. A block of the schedule $b \subseteq B$ is defined as a partial sequence that is scheduled consecutively without any idle time between jobs belonging to it.

The jobs of every block b are partitioned into two subsets: $Dec(b)$ and $Inc(b)$. If $j \in Dec(b)$ and the block is shifted earlier, the value of g_j either decreases or

does not change; on the other hand, if $j \in Inc(b)$ and the block is shifted earlier, the value of g_j either increases or does not change.

A complete schedule π has l blocks $\{1,2,\ldots,l\}$ $(1 \le b \le l; l \le |J|)$. In the processing of the algorithm we need to know just the extreme indices of the block, *first*(*b*) (the first job of *b*) and *last*(*b*) (the last job of *b*). The remaining jobs are easily found since they are directly related to the first and the last (See Fig. 1).

In the development of the algorithm we have to consider that the sum of convex functions in a convex set is convex, as can be seen in [10, p. 178] for instance. The result is that a sum of convex functions in a convex set has only a minimum point. In our case, as each job uses a convex function to define its contribution to the objective function, the block positions and, consequently, the schedule, are also convex functions of the completion times. Thus unconstrained minimisation techniques can handle properly the problem.

Observe that we are not violating any discrete property of scheduling problems, since we will just shift blocks trying to find the minimum cost position.

Let *z* be an indecision interval related to the optimum completion time of a job or set of jobs belonging to a block. For C_j a certain completion time with an indecision interval *z*, we can say that the optimal completion time C_j^* belongs to the interval $[z\alpha C_j, z(1-\alpha)C_j]$ where $0 < \alpha < 1$.

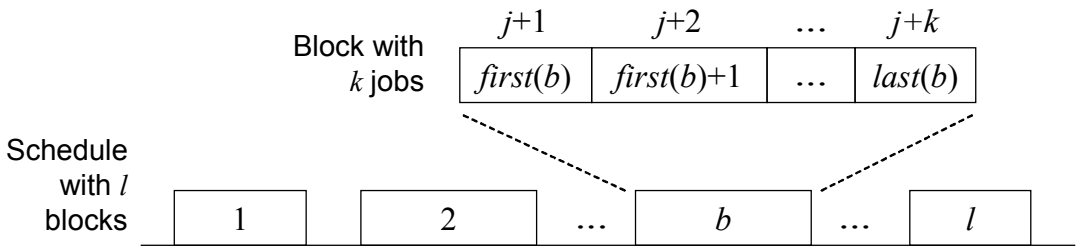

Fig. 1: Example of a schedule (partial or complete)

Lemma. If $g(x)$ is a convex function and $z^{[1]}$ is an initial indecision interval in which the minimum must lie, the golden section method, after m iterations, defines a smaller interval $z^{[m]}$ where the minimum point can be found.

Proof: The proof is a direct consequence of the method. The algorithmic definition of the method can be found in [3, pp. 270-272] for instance. □

We define $G(b) = \sum_{j \in b} g_j(C_j)$ as the objective function of the block for every block $b \in B$. Further we define $G(b^+)$ as the value of $G(b)$ if b were shifted earlier in a small value and $G(b^-)$ accordingly, for a small deferral of the block. If we consider that our indecision interval is defined by z, we can say that $G(b^+) = \sum_{j \in b} g_j(C_j - \alpha z)$ and $G(b^-) = \sum_{j \in b} g_j(C_j + (1-\alpha)z)$, where $0 < \alpha < 1$. In a practical point of view α is simply a factor that allows the right and left shifts of the block to be different while keeping the sum of both equal to z.

In the development of the algorithm we consider five possible regions in a convex function: type 1, 2, 3, 4, and 5, according to Fig. 2.

The algorithm takes into consideration the five possible regions of convex functions, and analyses the last two shifts. Let us suppose that the algorithm has performed shifts [n−1] and [n]. The algorithm performs the actions using the conclusions presented in table 1. The algorithm is described more formally below.

Let us suppose that the planning horizon starts at t_0. Without loss of generality, suppose that t_0=0. The algorithm starts scheduling the first job with minimum $g_1(C_1)$ If $C_1^* < p_1$, then, $C_1^{AIIT} = p_1$. Otherwise, we can define $C_1^{AIIT} = C_1^*$. Consider that a partial schedule has already been done until job j, π_j^{AIIT}, the j+1-th job can be scheduled according to one of the two possibilities: the first where the job can be scheduled with minimum cost and the other where it cannot.

Case 1 (guaranteed minimum cost): $C_j^{AIIT} + p_{j+1} \leq C_{j+1}^*$. We schedule j+1 such that $C_{j+1}^{AIIT} = C_{j+1}^*$. A new block is created with job j+1.

Case 2 (no guaranteed minimum cost): $C_j^{AIIT} + p_{j+1} > C_{j+1}^*$. We schedule j+1 such that $C_{j+1}^{AIIT} = C_j^{AIIT} + p_{j+1}$. j+1 is scheduled to start at the end of block l.

The algorithm will maintain the following key property: $G(l) \leq G(l^+)$, that is, an additional shift either maintains or increases the value of the objective function. If after the inclusion of j+1 in the scheduling π_j^{AIIT} the key property is maintained, we do not take any action. On the other hand, if the key property is not maintained, block l is moved earlier while at least one of the following conditions do not hold:

Condition A: $e_{first(l)} = 0$. It can happen only if $l = 1$. In this case, block l cannot be moved earlier because it is starting at the earliest starting time;

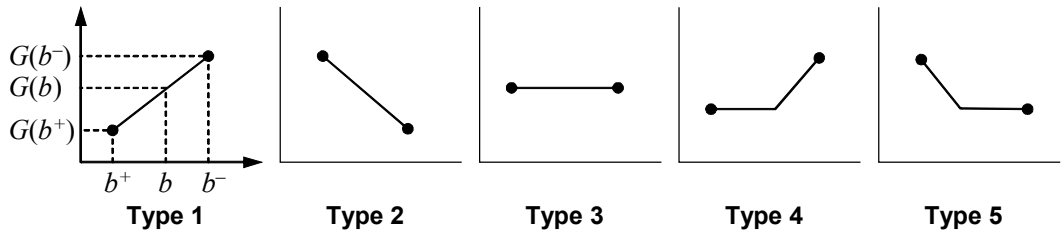

Fig. 2: Possible regions of a convex function

	Point		Conclusion
	$[n-1]$	$[n]$	
Region	type 1	type 1	Shift earlier once again
	type 1	type 2	The point of minimum cost is between $[n-1]$ and $[n]$
	type 1	type 3	The point of minimum cost is b of $[n]$
	type 1	type 4	The point of minimum cost is b of $[n]$
	type 1	type 5	The point of minimum cost is b of $[n]$

Table 1: Possible region relationships of two consecutive iterations of the algorithm

Condition B: The inequality $G(l) \le G(l^+)$ becomes true; the algorithm assumes that the block is in a region of type 2, 3, 4, or 5. In this case, if the block is in a region of type 2, it means that a smaller shift might decrease the value of the objective function. In other words, two consecutive iterations, [n−1] and [n], have evaluated region as being of type 1 and 2 respectively. If it occurs, we know the block has to be delayed. Since every job in the block is related to the others, if we find the optimal position of *first*(l), we will find the optimal position of the entire

block. The search will start in the interval $[C^{[n]}_{first(l)}, C^{[n-1]}_{first(l)}]$ and will find a new interval that is either smaller or equal to z (the previously defined precision).

Observe that for shifts, blocks l and $l-1$ can be merged $(l \leftarrow l \cup l-1)$ several times. The merge does not claim changes in the described algorithm, since the merged block has the same treatment as the initial block l.

Let $\Delta^{[n]} = \min_{j \in Dec(b)} (C_j - C_j^*)$ be the shift size, after n shifts, that made the second condition become true, positioning the block in a region of type 2. The indecision interval $[C^{[n]}_{first(l)}, C^{[n-1]}_{first(l)}]$, with $z^{[1]} = C^{[n-1]}_{first(l)} - C^{[n]}_{first(l)}$, contains the point of minimum cost of l. The utilisation of the one-dimensional search procedure over the interval $z^{[1]}$ determines a smaller interval where the minimum value of $G(l)$ lies. The step came up with an optimal shift Δ^* that replaces shift $\Delta^{[n]}$.

We use a one-dimensional procedure known as golden section method [10, pp. 199-200] to find the minimum of the function within some tolerance in the solution space. In the golden section method, after m iterations, the indecision interval is reduced to $z^{[m]} = (2/(1+\sqrt{5}))^{m-1} z^{[1]}$. Suppose the final indecision interval is $z \geq z^{[m]}$. It will be necessary

$$m \leq 1 + \frac{\ln(z / z^{[1]})}{\ln(2/(1+\sqrt{5}))}$$

interactions to the procedure be concluded.

After the definition of Δ^* for j+1, if it is the case, the resulting schedule is π_{j+1}^{AIIT}. The algorithm carries out successive applications of the described procedure to form schedules $\pi_2^{AIIT}, \pi_3^{AIIT}, ..., \pi_{|J|}^{AIIT}$.

We can consider that for the partial schedule π_j^{AIIT}, the minimum values of the objective functions of the blocks belong to indecision intervals z. These indecision intervals represent an acceptable precision for the optimum position of the blocks. In a practical point of view, z is easily justified since completion times generally are not penalised in a continuous basis. In fact, completion times are penalised according to time windows. This means that whichever the completion time is, the penalisation is the same, since it belongs to the same time window (e.g. a day can be considered a time window since a delivery delay of either 7.5 or 7.7 days is virtually the same). Taking these observations into account, the following theorem can be stated:

Theorem. π_j^{AIIT} is a schedule with minimum value of $g(\pi_j) = \sum_{i=1}^{j} g_i(C_i)$ within a tolerance z in the solution space, among all possible partial scheduling for the first j jobs.

Proof: As the induction hypothesis, we consider that in j the theorem is true, that is, $g(\pi_j^{AIIT}) = \min_{\pi_j \subseteq \Pi_j} g(\pi_j)$, where Π_j is the set of all possible schedules for the first j jobs. For the empty schedule π_0^{AIIT}, the theorem is true. Thus, we have to prove that the theorem is also true for j+1. Let C_k and C_k^{AIIT} be completion times of the job k in π_j and π_j^{AIIT}, respectively. We can consider two cases: In the first, the job is

completed in its target time, and in the second, it is not:

Case 1. $C_j^{AIIT} + p_{j+1} \leq C_{j+1}^*$. This case yields the algorithm schedules *j*+1 such that $C_{j+1}^{AIIT} = C_{j+1}^*$, and thus, for π_{j+1}^{AIIT} the objective function is $g(\pi_{j+1}^{AIIT}) = g(\pi_j^{AIIT}) + g_{j+1}(C_{j+1}^*)$. Taking into consideration that $C_{j+1}^* = \arg\min g_{j+1}(C_{j+1})$ by definition and $g(\pi_j^{AIIT}) = \min_{\pi_j \subseteq \Pi_j} g(\pi_j)$ by the induction hypothesis, we can say surely that $g(\pi_{j+1}^{AIIT}) \leq g(\pi_{j+1})$. Therefore the theorem is true for this case.

Case 2. $C_j^{AIIT} + p_{j+1} > C_{j+1}^*$. As it is not possible $C_{j+1}^{AIIT} = C_{j+1}^*$ to occur anymore, the algorithm schedules initially *j*+1 in a such way that $C_{j+1}^{AIIT} = C_j^{AIIT} + p_{j+1}$, and evaluate $e_{first(l)}^{AIIT}$ and the relationship between *G*(*l*) and $G(l^+)$. *j*+1 is inserted into the block *l*. Considering the induction hypothesis and the convexity of $g_{j+1}(C_{j+1})$, we can say that $g_{j+1}(C_{j+1}^{AIIT}) \leq g_{j+1}(C_{j+1}^{AIIT} + \varepsilon)$ where $C_{j+1}^* < C_{j+1}^{AIIT} \leq C_{j+1}^{AIIT} + \varepsilon$. With this consideration, we know that $g(\pi_{j+1})$ can be decreased only if block *l* is shifted earlier Δ units. Without loss of generality, let us suppose that shifts do not merge blocks *l* and *l*–1.

If in any moment $e_{first(1)}^{AIIT} = 0$, the algorithm does not make any further shift because there is no shift Δ that can improve the objective function. Therefore the theorem proves the optimality of the algorithm. On the other hand, if in the shifts $e_{first(1)}^{AIIT} \neq 0$, the algorithm shift block *l* earlier until at least one of the two following subcases happen:

Subcase 2a. The inequality $G(l) \leq G(l^+)$ becomes true, and an analysis of the objective function (in shift [n]) indicates the block is in a region of type 3, 4, or 5. Due to the convexity of $G(l) = \sum_{j \in l} g_j(C_j)$, the inequality suffices to guarantee that the minimum point belongs to an interval z since the sum of right and left shift widths is at most z. This completes the proof for this subcase.

Subcase 2b. The inequality $G(l) \leq G(l^+)$ becomes true, and an analysis of the objective function (in shift [n]) indicates that the block is in a region of type 2. Considering the algorithm's progression, in the shift [n−1] the region was of type 1, and thus, we can say that the objective function's derivative inverted the signal between the shifts [n−1] and [n]. This case demands a search (that will be carried out with the golden section method) in the region belonging to the interval $[C_{first(l)}^{AIIT[n]}, C_{first(l)}^{AIIT[n-1]}]$. Taking the lemma into account we can say that a number m of iterations will happen until $z^{[m]} \leq z$. The proof is now completed. □

4. Computational implementation and related questions

Some operations of the algorithm is implemented using heaps, that according to [12, p. 33], can be defined as "an abstract data structure consisting of a collection of items, each with an associated real-valued key".

For every block b a heap $P(b)$ is maintained. It defines how much each one can be shifted. Every heap must maintain two values for each job: the first is the index of the job belonging to $Dec(b)$; the second is a value equivalent to $C_j - C_j^*$ (the maximum amount of time that j can be shifted while its tardiness decreases) for every job $j \in Dec(b)$, hereafter denoted as the key of the heap.

The algorithm is implemented considering eight operations. For n the number of items in the heap and considering that heaps are represented by binary trees, the eight operations with their worst computation times - the first 6 found in [12, chap. 3] - are described below:

Operation 1. MakeHeap(P). $O(1)$. Create a new empty heap denoted P;

Operation 2. FindMin(P). $O(1)$. Find an item of minimum key in heap P returning its key without removing the item from P;

Operation 3. DeleteMin(P). $O(\log n)$. Find an item of minimum key in heap P. Presents itself being deleted from P;

Operation 4. Insert(j,x,P). $O(1) + O(\log n) = O(\log n)$. Insert an item j with key x in heap P;

Operation 5. Merge(P_1,P_2). $O(\log(n_1 + n_2)) = O(\log n)$. Merge heaps P_1 and P_2 (with n_1 and n_2 items) of disjointed items into a new heap that becomes P_1. P_2 becomes empty;

Operation 6. AddToAllKeys(x,P). $O(1)$. Add x to all keys of the items belonging to P;

Operation 7: DefineRegionType($-\Delta$). $O(n)$. Computes $G(l)$ and $G(l^+)$ for a shift of Δ and evaluates the type of the region;

Operation 8: GoldenSection(a,b,f,z). $O(\log(d_{\max} / z))$. The golden section method needs at most $m = 1 + \log(z/\Delta)/\log(2/(1+\sqrt{5})) = O(\log(\Delta / z))$ (where Δ is a shift) iterations until the indecision value be smaller or equal to z. The largest possible shift is $d_{\max}$; farthest point where the algorithm

would place a job. It uses the golden section method to define an interval where the position of minimum cost of the block lies. Given a function f, and an interval $[a,b]$ where the search must be performed, the operation finds, with a precision of z, the position that keeps the block at minimum cost.

Subsequently we present the pseudocode of the AIIT described previously. The procedure is called InsertIdleness, according to Fig. 3 and has an auxiliary procedure denoted ShiftEarlier, according to Fig. 4.

The computational code of the golden section method is quite common and can be found in many sources such as [11, pp. 390-395] for example.

In the pseudocode we did not introduce a definition for α because it depends on the characteristics of the problem being analysed. Furthermore, the computation of α does not bring additional troubles neither has it a significant impact in the computational time of the algorithm.

procedure **InsertIdleness** (π_0,z)
begin
for j:=1 **to** n **do**
$C_j^* := \arg\min g_j(C_j)$; l:=1; *first*(1):=1; *last*(1):=1;
if $C_1^* < p_1$ **then**
C_1:=p_1;
else
$C_1 := C_1^*$;
$G_f(1) := g_1(C_1)$; MakeHeap(P(1));
for j:=1 **to** n−1 **do**
if $C_j + p_{j+1} < C_{j+1}^*$ **then**
begin
l:=l+1; $C_{j+1} := C_{j+1}^*$; *first*(l):=j+1; *last*(l):=j+1; $G_f(l) := g_{j+1}(C_{j+1})$; MakeHeap($P$($l$));
end;
else
begin
last(l):=j+1; C_{j+1}:=C_j+p_{j+1}; Insert$(j+1, C_{j+1} - C_{j+1}^*, P(l))$; $G_f(l) := G_f(l) + g_{j+1}(C_{j+1})$;
while $G_v(l) > G_v(l^+)$ **and** $C_{first(l)} - p_{first(l)} \neq 0$ **do**
ShiftEarlier;
endwhile;
end;
end.

Fig. 3: Pseudocode for insertion of idle time in a fixed sequence - job costs represented as convex functions of the completion times

procedure <u>ShiftEarlier</u>
begin
Δ1:=FindMin($P(l)$);
if l=1 **then**
$\Delta 2 := C_1 - p_1$;
else
$\Delta 2 := C_{first(l)} - p_{first(l)} - C_{last(l-1)}$;
Δ:=min{Δ1,Δ2};
NewRegion:=DefineRegionType(−Δ);
if NewRegion=Type2 **then**
Δ:=GoldenSection $(C_{first(l)} - \Delta, C_{first(l)}, G_f(l), z)$;
AddToAllKeys $(-\Delta, P(l))$; $C_{first(l)} := C_{first(l)} - \Delta$; $C_{last(l)} := C_{last(l)} - \Delta$;
while FindMin($P(l)$)=0 **do**
k:=DeleteMin($P(l)$);
end de while;
if l>1 **and** $C_{last(l-1)} = C_{last(l)} - p_{last(l)}$ **then**
begin
Merge($P(l-1)$,$P(l)$); $last(l-1):=last(l)$; $G_f(l-1) := G_f(l-1) + G_f(l)$; l:=l−1;
end;
end;

Fig. 4: Pseudocode of the procedure ShiftEarlier

The reader can verify that $G(l)$ either is used as $G_f(l)$ or as $G_v(l)$. The difference between them lies in the fact that the first represents the function $G(l)$ and the second the numerical value of $G(l)$. Although there is no elucidation, $g_j(C_j)$ follows the same standard in the assignment operations. If $x := G_f(l) + g_j(C_j)$ we want to mean that the function $g_j(C_j)$ is being added to $G_f(l)$, while if $x := G_v(l) + g_j(C_j)$ we want to mean that the value of $g_j(C_j)$ is being added to $G_v(l)$.

Proposition. The computational time of the algorithm is $O(|J| \max\{|J|, \log(d_{\max}/z)\})$, where $d_{\max}$ is the largest due date, $|J|$ is the number of jobs and z is the precision which the search defines the optimal region.

Proof: The time for the golden section method, as previously seen, is $O(\log(d_{\max}/z))$ and the time for the operation DefineRegionType(x) is $O(j)$ considering that the block contains j jobs. In the worst case, these operations will be used as many times as the ShiftEarlier procedure does. The DeleteMin procedure runs in $O(\log j)$ for j items in the heap. In the worst case the ShiftEarlier procedure will be used $|J|-1$ times. Thus, the total time can be defined as

$$\sum_{j=1}^{|J|-1} \{\log(d_{\max}/z) + j\} = (|J|-1)\log(d_{\max}/z) + \frac{(|J|-1)|J|}{2}$$
$$= O(|J| \max\{|J|, \log(d_{\max}/z)\}).$$

If the ShiftEarlier procedure is used $|J|-1$ times, the Insert operation also does. Thus, the combined running time for the entire procedure can be defined as $O(|J|\log|J|)+O(|J|\max\{|J|,\log(d_{\max}/z)\})=O(|J|\max\{|J|,\log(d_{\max}/z)\})$, which means the proposition is true. □

The proposition shows that the algorithm has running time bounded by a pseudo-polynomial function in the instance size.

5. Concluding Remarks

This paper considers an algorithm that can be used in a broad class of objective functions for scheduling with nonlinear earliness and tardiness costs. The relevance of the work could be greater if there were more studies about scheduling with nonlinear cost functions. The very importance of nonlinear costs in scheduling is not properly highlighted yet; besides the large number of sequencing and scheduling studies, there is a lack of research considering nonlinearities. We can mention an example considering that costs of perishable items (earliness costs) are far better modelled by nonlinear functions. This is also true for the tardiness case; the greater the tardiness the less sustainable the situation between producer and client, showing clearly the importance of nonlinear models. Of course this additional complexity transforms a hard problem into one even harder, however, nonlinear functions such as exponential or quadratic perhaps could bring substantial improvements in the

modelling of real problems. We think this paper is a small effort towards this direction.

The subject approached here seems to have interesting deployments in the development of procedures that use it. A piece of research considering sequencing and scheduling at the same time can be interesting, specially if we take into account that idle time insertion algorithms can give valuable insights into sequencing algorithms.

References

[1] B. Alidaee, Single machine scheduling with nonlinear cost functions, Computers and Operations Research 18 (1991) 317-322.

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

[3] M.S. Bazaraa, H.D. Sherali, C.M. Shetty, Nonlinear programming: theory and algorithms, 2nd. ed., Wiley, New York, 1993.

[4] R.L. Carraway, R.J. Chambers, T.L. Morin, H. Moskowitz, Single machine sequencing with nonlinear multicriteria cost functions: an application of generalized Dynamic Programming, Computers and Operations Research 19 (1992) 69-77.

[5] E.C. Colin, T. Shimizu, Algoritmo de programação de máquinas individuais com penalidades distintas de adiantamento e atraso, Pesquisa Operacional 20 (2000) 15-26.

[6] J.S. Davis, J.J. Kanet, Single-machine scheduling with early and tardy completions costs, Naval Research Logistics 40 (1993) 85-101.

[7] T.D. Fry, R.D. Armstrong, J.H. Blackstone, Minimizing weighted absolute deviation in single machine scheduling, IIE Transactions 19 (1987) 445-450.

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

[9] J.J. Kanet, V. Sridharan, Scheduling with inserted idle time: problem taxonomy and literature review, Operations Research 48 (2000) 99-110.

[10] D.G. Luenberger, Linear and nonlinear programming, 2nd. ed., Addison-Wesley, Reading, 1984

[11] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical recipes in FORTRAN: the art of science computing, 2nd. ed., Cambridge University Press, Cambridge, 1992.

[12] R.E. Tarjan, Data structures and network algorithms, Society for Industrial and Applied Mathematics, Philadelphia, 1983.