MILP formulation and polynomial time algorithm for an aircraft scheduling problem*

Alexandre M. Bayen† Jiawei Zhang‡ Claire J. Tomlin§ Yinyu Ye¶

March 4, 2003

Submitted to the IEEE CDC 2003.

Abstract

This article presents a polynomial time algorithm used for solving a MILP formulation of a scheduling problem applicable to Air Traffic Control. We first relate the general MILP (which we believe to be NP-Hard) to the Air Traffic Control problem. This MILP can be solved with CPLEX without guarantee on the running time. We show that a specific case of the Air Traffic Control problem, which is of interest on its own right, admits an exact polynomial-time algorithm. Our method reduces the feasibility of the MILP to a single machine scheduling problem, and embeds the solution algorithm in a bisection algorithm. We establish the polynomial complexity of our bisection algorithm by proving an algebraic property of the optimal solution. We compare the running times of CPLEX and our algorithm for 1800 cases. The results show numerical evidence of the guaranteed running time of our algorithm, by contrast with CPLEX's whose average performance is good, but also shows a significant number of instances with unpredictably large computational time. We perform 8100 additional runs of our algorithm, to numerically confirm the predicted worst case running time of our algorithm up to 100 aircraft.

Keywords: MILP, scheduling, polynomial-time algorithm.

*Research supported by NASA under Grant NCC 2-5422, by ONR under MURI contract N00014-02-1-0720, by DARPA under the Software Enabled Control Program (AFRL contract F33615-99-C-3014), and by a Graduate Fellowship of the Délégation Générale pour l'Armement (France).

†Ph.D. Student, IEEE Student Member, corresponding author. Aeronautics and Astronautics, Durand 028, Stanford University, Stanford CA, 94305-4035. Tel: (650)498-0530, Fax: (650)723-3738. bayen@stanford.edu

‡Ph.D. Student, Management Science and Engineering, Stanford University.

§Assistant Professor, Aeronautics and Astronautics, Courtesy Assistant Professor, Electrical Engineering; Director, Hybrid Systems Laboratory, Stanford University.

¶Professor, Management Science and Engineering; Director, Computational Optimization Laboratory, Stanford University.

1 Introduction

1.1 Origins of the problem

Motivated by the growth of air traffic since the 1940s, the *Federal Aviation Administration* (FAA) achieved a semi-automated *Air Traffic Control* (ATC) system, based on a combination of radar and computer technology, in the mid-1970s. This system has been constantly upgraded since, leading to the *Center-TRACON Automation System* (CTAS), a hardware and software tool suite aimed at helping the Air Traffic Controllers manage the increasing complexity of air traffic flow in the vicinity of large airports. Besides reducing the stress and workload of the Controllers, this partial automation of the system also contributed to a more efficient traffic flow management which benefits the passengers and the airlines, by reducing the delays and improving safety. The functionalities covered by this automation include monitoring, alerts, advisories, planning, displays. However, other key functionalities in today's system are only partially automated, and not optimized. The most fundamental of them is conflict avoidance, which is currently performed manually. Arrival planning, which is another crucial task of ATC in the vicinity of airports is currently assisted by the NASA-developed software CTAS [14], which helps controllers generate optimal *flight plans* and trajectories for traffic incoming into airports.

A flight plan is a set of *waypoints* (reference points defined precisely in the airspace), which the aircraft are expected to follow. Even though in low traffic density regions, aircraft might fly off these flight plans to benefit from faster routes (because of winds, for example), when this airspace becomes congested, aircraft will follow arrivals for up to 200 nautical miles (nm) from the destination airport. Arrivals aid the Air Traffic Controller in the problem of flow *metering*, or delivering a prescribed number of aircraft per unit time to the airport runway, as the routes can be viewed as tracks which the aircraft follow closely with minor deviations until they reach the arrival airport. An Air Traffic Controller can thus regulate the flow by adjusting the flight plans of individual aircraft, according to procedures (called *playbooks*) which have been established over time to meet the acceptance rates at airports. Adjusting the flight plan of an aircraft consists first of assigning the aircraft to an arrival (as shown in Figure 1 right). Once this assignment is done, smaller adjustments can be performed, to achieve prescribed arrival times precisely. Two examples of maneuvers used for such adjustments are shown in Figure 2.

In the current system, Air Traffic Controllers tend to use only a subset of the maneuvers available to them while metering the flow into the airports: for example, it is often easier not to change the order of arrival of the aircraft into an airport, despite the fact that it might reduce the delays. In [4], we study the feasibility of automatically generating these flight plan adjustments, in order to meter flow in real time, even when the system is operating at maximal capacity. This study takes into account the full set of maneuvers (see [3, 4, 18] for a more detailed description of the maneuvers) available to ATC, and does not make any a priori assumptions regarding their use — as is done today by Air Traffic Controllers. The main result of [4] is an algorithm which maps the set of all possible adjustments of all aircraft to a *Mixed Integer Linear Program*

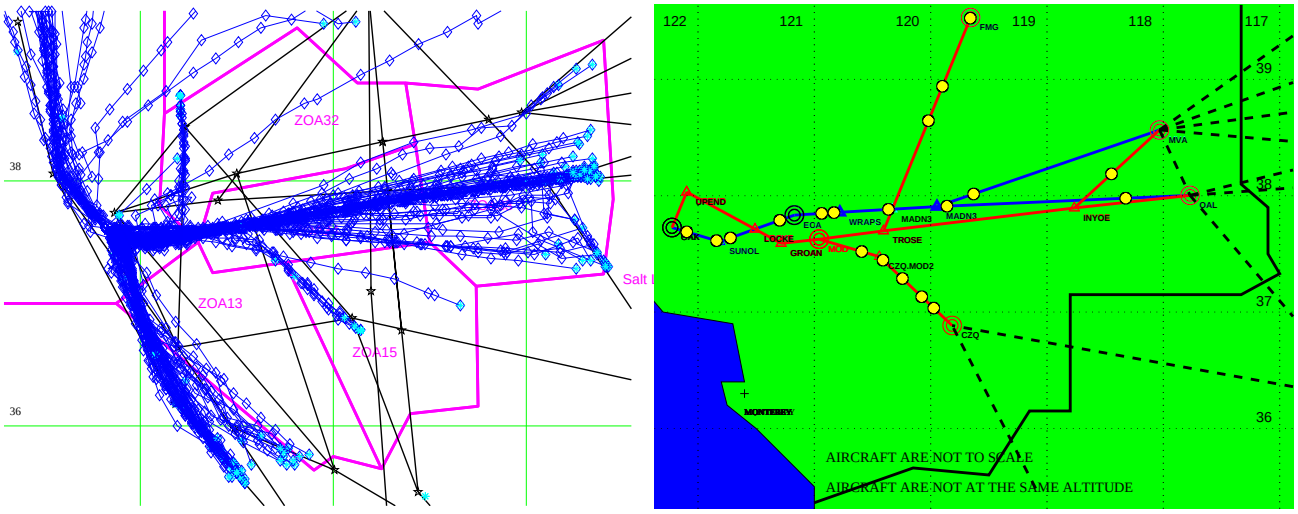

Figure 1: **Left:** Overlay of trajectories merging into San Francisco (11 hours of traffic). The data comes from ETMS and FACET [8]. The algorithm presented in this article can be used for scheduling this incoming flow. **Right:** Two examples of arrival into the Oakland airport (Madwin 3 and Locke 1).

(MILP). In [4], we propose an implementation of our algorithm, which takes ETMS[1] data as input, and generates a schedule when the constraints imposed by the airport are feasible. Practical implementations of advisory tools for arrival planning require several points to be addressed. For a half million line code like CTAS, architecture and software design techniques become a key issue to address [19]. But beyond the difficulties of software engineering, the complexity of the underlying mathematical problem (in the present case a hybrid controller synthesis problem) is a bottleneck that engineering skills cannot overcome without proper algorithmic treatment of its formulation. For the current work, the computational time bottleneck of the algorithm [4] is the MILP formulation, which in our preliminary implementation, was solved with CPLEX [16] (see Figure 3). The conclusion of [4] was that despite the very good average performance of CPLEX, unpredictable cases, in which solving the MILP takes an exponentially large amount of time, occur in practice. This behavior is undesired for an online implementation of the algorithm, because the user might have to wait an unacceptable amount of time before getting an answer. This article proposes an algorithm for which we provide an upper bound on the running time of the MILP solution, and which is therefore guaranteed to find the exact optimal solution in a time which is predictable.

[1] The *Enhanced Traffic Management System* database contains all flight plan information for flights in the National Airspace System (NAS). Data are collected from the entire population of flights in the NAS with filed flight plans. ETMS data is sent from the Volpe National Transportation System Center (VNTSC) to registered participants via the *Aircraft Situation Display to Industry* (ASDI) electronic file server. The FAA uses these data to monitor the effectiveness of its National Route Program, in which the user community is offered flexible, cost-effective routing options as an alternative to published ATC preferred routes.

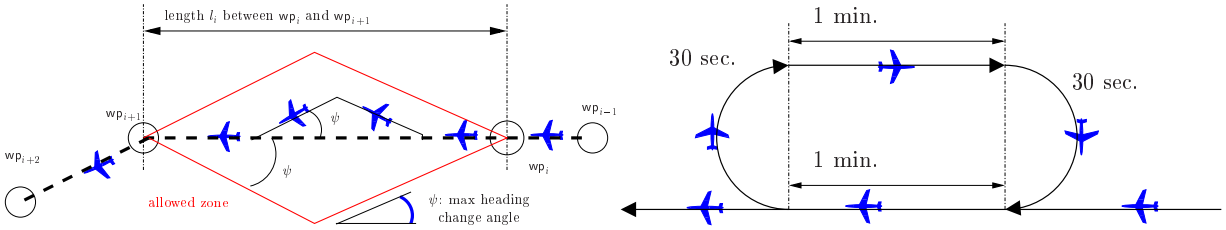

Figure 2: Two examples of maneuvers corresponding to arrival time adjustment. **Left:** Deviation ψ from flight plan using a *Vector For Spacing* (VFS) available in a given sector. More precise models are also available in [13]. **Right:** *Holding Pattern* (HP). The prescribed "time to lose" is given to the aircraft in minutes: one minute in each straight portion and 30 seconds in each half circle. For this scenario, $T_{\mathsf{hp}} = 1\,\text{min.} + 30\text{ sec.} + 1\,\text{min.} + 30\text{ sec.} = 3\,\text{min.}$

1.2 Complexity of the problem, related work, organization of this article

Mixed integer linear programming [7] is a powerful mathematical formulation that extends linear programming to problems with both continuous and integer variables. It appears naturally in various fields where these two types of variables coexist, for example processing or chemical engineering [20, 15]. It enables inclusion of computational logic [24] into optimization problems, and provides an excellent tool for multi-vehicle or conflict avoidance problems [21, 22] and discrete time hybrid systems [6, 5].

Integer programming (and therefore MILP) in the general case is NP-Hard [17, 23]. Famous examples of NP-Hard problems, which can be posed as integer programs, include facility location, traveling salesman, knapsack, bin packing [23]. Certain problems which can be formulated as integer programs (for example max-flow, min-cut, maximum matching, minimum spanning tree) are polynomial-time solvable [1]. But a general integer programming optimization software such as CPLEX [16] cannot differentiate between polynomial-time solvable problems and NP-Hard problems a priori, and might require exponential time to find the solution of a polynomial-time solvable problem. We experienced this fact in [4]; this article shows that a specific case of the problem shown in [4] is polynomial-time solvable by deriving an exact polynomial-time algorithm.

This article is organized as follows. In section 2, we present the general formulation of the aircraft routing-sequencing problem, and the specific case of interest. We identify a closed form solution for the particular case in which the order of arrival of the aircraft is fixed in advance. In section 3, we use this closed form solution as a stopping condition for a bisection algorithm used to compute the exact solution of the MILP. The bisection algorithm relies on a feasibility algorithm which is a modification of a known single processor scheduling algorithm. This algorithm is derived in the Appendix, with a numerical example to illustrate its structure. Section 4 presents a numerical implementation of this algorithm, and extensive comparisons with CPLEX. The bounds on running time derived for our algorithm are verified in practice and it is shown to perform much better than CPLEX. Section 5 outlines the current state of our research for the general version of the problem, as well as preliminary results for this general problem.

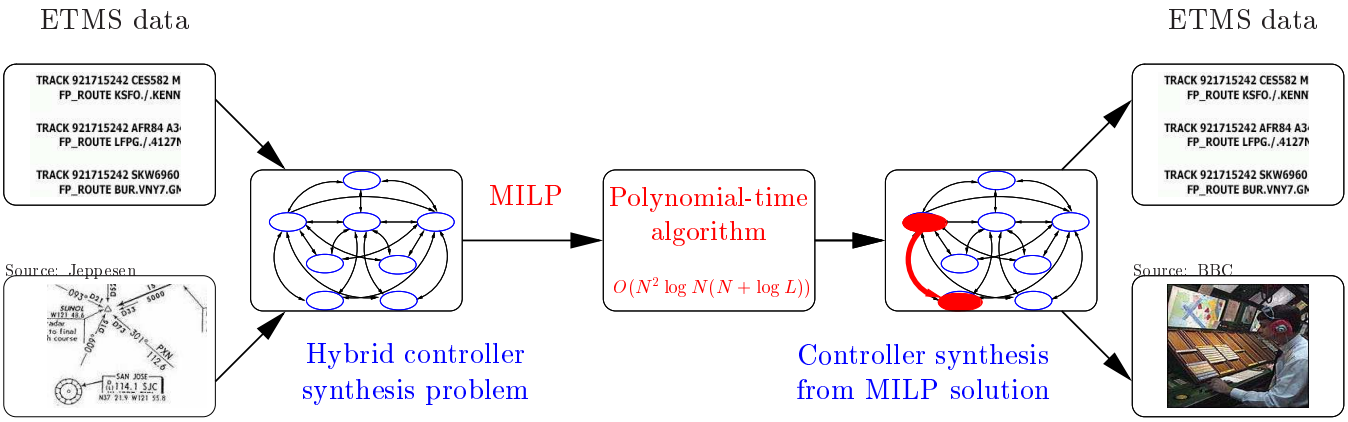

Figure 3: Block diagram representation of the embedding of the algorithm of this article within the algorithm presented in [4]. The input of the algorithm of [4] is the set of flight parameters of the aircraft (from ETMS data [8]), as well as the known structure of the arrivals. The resulting hybrid systems controller synthesis problem is posed as a MILP, which is solved by the algorithm presented in this article. The output, which is an execution of the hybrid system [4], is translated in ETMS data for simulation or set of ATC commands for advisory purposes.

2 Problem formulation and preliminary remarks

2.1 Problem formulation

We consider N aircraft converging to a single airport. Aircraft are indexed by $i \in \{1, \cdots N\}$. As a result of ATC actuation, the set of arrival times for aircraft i is a union of intervals called $S_i := \bigcup_{k=1}^{n_i} [a_{i,k}, b_{i,k}]$. For each i, n_i is the number of intervals of time in which aircraft i can arrive. These intervals encode a set of maneuvers which are relatively easy to implement for ATC. A key problem for an Air Traffic Controller is to estimate the largest time spacing achievable with this set of maneuvers, in order to determine if the use of large maneuvers is necessary (significant flight plan changes or large numbers of holding patterns at the direct vicinity of airports). This question translates into the following optimization problem: given $\{S_i\}_{i \in \{1, \cdots N\}}$, we want to find a N-tuplet $(t_1, \cdots, t_N) \in \prod_{i=1}^{N} S_i$ of feasible arrival times for the N aircraft maximizing the minimum spacing Δ^* between any two aircraft:

$$\begin{array}{lll} \textbf{Maximize:} & \Delta & \\ \textbf{Subject to:} & t_i \in \bigcup_{k=1}^{n_i} [a_{i,k}, b_{i,k}] & \forall i \in \{1, \cdots N\} \\ & |t_i - t_j| \geq \Delta & \forall (i,j) \in \{1, \cdots N\}^2 \end{array} \quad (1)$$

In this form, (1) is not a linear program (it is not even convex). It is possible to rewrite (1) as a MILP, using boolean variables c_{ik} and d_{ik} to express the constraints (see [4]):

$$
\begin{array}{lll}
\textbf{Maximize:} & \Delta & \\
\textbf{Subject to:} & t_i \geq a_{i,1} & \forall i \in \{1, \cdots, N\} \\
& t_i \leq b_{i,n_i} & \forall i \in \{1, \cdots, N\} \\
& t_i \geq a_{i,k+1} - D d_{ik} & \forall i \in \{1, \cdots, N\}, \forall k \in \{1, \cdots, n_i - 1\} \\
& t_i \leq b_{i,k} + D(1 - d_{ik}) & \forall i \in \{1, \cdots, N\}, \forall k \in \{1, \cdots, n_i - 1\} \\
& d_{ik} \in \{0, 1\} & \forall i \in \{1, \cdots, N\}, \forall k \in \{1, \cdots, n_i - 1\} \\
& t_i - t_j \geq \Delta - C c_{ij} & \forall (i,j) \in \{1, \cdots, N\}^2, \text{ s.t. } i > j \\
& t_i - t_j \leq -\Delta + C(1 - c_{ij}) & \forall (i,j) \in \{1, \cdots, N\}^2, \text{ s.t. } i > j \\
& c_{ij} \in \{0, 1\} & \forall (i,j) \in \{1, \cdots, N\}^2, \text{ s.t. } i > j
\end{array}
\tag{2}
$$

In (2), C and D are "large" constants. Large means that they have to satisfy: $C \geq 2(\max_{j=1}^{N} b_{j,N_j} - \min_{i=1}^{N} a_{i,1})$ and $D \geq \max_{i=1}^{N} \max\{a_{i,n_i} - a_{i,1}, b_{i,n_i} - b_{i,1}\}$. We do not know if this problem in general is polynomial-time solvable or not. Given the similarity with known NP-Hard scheduling problems (for example, if the separation of the aircraft depends on the aircraft type), we believe that this problem is NP-Hard, though we were not able to prove this yet.

For the present work, we focus on the one interval case ($n_i = 1$ for all i), and show that this problem is polynomial-time solvable. In the one interval case, d_{ij} disappears, and the problem becomes:

$$
\begin{array}{lll}
\textbf{Maximize:} & \Delta & \\
\textbf{Subject to:} & t_i \geq a_{i,1} & \forall i \in \{1, \cdots, N\} \\
& t_i \leq b_{i,1} & \forall i \in \{1, \cdots, N\} \\
& t_i - t_j \geq \Delta - C c_{ij} & \forall (i,j) \in \{1, \cdots, N\}^2, \text{ s.t. } i > j \\
& t_i - t_j \leq -\Delta + C(1 - c_{ij}) & \forall (i,j) \in \{1, \cdots, N\}^2, \text{ s.t. } i > j \\
& c_{ij} \in \{0, 1\} & \forall (i,j) \in \{1, \cdots, N\}^2, \text{ s.t. } i > j
\end{array}
\tag{3}
$$

2.2 Closed form solution for fixed arrival order

In order to solve (3), we need to identify which constraints are active in the optimal solution: in the present case, they determine the optimum of the problem analytically. For this, we fix the order of arrival at an arbitrary setting, which enables us to find the active constraints. The result obtained for this arbitrary order will be used for solving the general problem (3). Theorem 1 is thus a building block of the full algorithm, which is presented in Section 3. Lemma 1 below is used to prove Theorem 1.

Thus, assume for this section only that the order of arrival of the aircraft is known a priori. Without loss of generality, assume that the aircraft have been labeled in this order (aircraft 1 arrives first, aircraft 2

arrives second, $\cdots$). Then, problem (3) reduces to

$$\begin{array}{lll} \textbf{Maximize:} & \Delta & \\ \textbf{Subject to:} & t_i - t_{i-1} \geq \Delta & i \in \{2, \cdots, N\} \\ & t_i \geq a_{i,1} & i \in \{1, \cdots, N\} \\ & t_i \leq b_{i,1} & i \in \{1, \cdots, N\} \end{array} \tag{4}$$

Lemma 1. *The optimal solution Δ^* of (4) satisfies $\Delta^* \leq m$, where*

$$m = \min_{(i,j) \in \{1,\cdots,N\}^2,\ i<j} \left(\frac{b_{j,1} - a_{i,1}}{j-i} \right) \tag{5}$$

Proof — Call $(k,l) = \arg\min_{(i,j)\in\{1,\cdots,N\}^2,\ i<j} \left(\frac{b_{j,1}-a_{i,1}}{j-i}\right)$, i.e. one pair of aircraft such that $\left(\frac{b_{l,1}-a_{k,1}}{l-k}\right) = m$ and $k < l$. Suppose that $\Delta^* > m$. Then $(l-k)m < (l-k)\Delta^* \leq b_{l,1} - a_{k,1} = (l-k)m$. The first inequality is by assumption. The second by definition of Δ^*: the best spacing between aircraft k and l is bounded by the mean. The last equality is by definition of m. This inequality cannot be true. Therefore $\Delta^* \leq m$. □

Theorem 1. *It is possible to construct a sequence $(t_1, \cdots, t_N)$ such that $(m, t_1, \cdots, t_N)$ satisfies (4), which proves that*

$$\Delta^* = \min_{(i,j) \in \{1,\cdots,N\}^2,\ i<j} \left(\frac{b_{j,1} - a_{i,1}}{j-i} \right) \tag{6}$$

Proof — We prove this theorem by induction on N. For $N = 2$, the solution is trivial. For $N = 3$, we see that the following solution achieves the optimal:

$$\begin{array}{lll} t_1 & = a_{1,1} & \\ t_2 & = (b_{3,1} + a_{1,1})/2 & \text{if } (b_{3,1} + a_{1,1})/2 \in [a_{2,1}, b_{2,1}] \\ & = a_{2,1} & \text{if } (b_{3,1} + a_{1,1})/2 < a_{2,1} \\ & = b_{2,1} & \text{if } (b_{3,1} + a_{1,1})/2 > b_{2,1} \\ t_3 & = b_{3,1} & \end{array} \tag{7}$$

Suppose now that we have proved the property up to N, i.e. that (6) is the solution of (4) for N aircraft. We now prove that (6) is the solution of (4) for $N+1$ aircraft. Call $m(k,l)$ the following quantity, with k and l both in $\{1, \cdots, N+1\}$, and $k < l$:

$$m(k,l) := \min_{(i,j) \in \{k,\cdots,l\}^2,\ i<j} \left(\frac{b_{j,1} - a_{i,1}}{j-i} \right) \tag{8}$$

We obviously have $m(1,N+1) = \min\left\{m(1,N), \min_{i\in\{1,\cdots,N\}}\left(\frac{b_{N+1,1}-a_{i,1}}{N+1-i}\right)\right\}$. We now investigate which of these terms achieves the min for $m(1,N+1)$.

Case 1: $m(1,N+1) = m(1,N)$. We thus have $m(1,N) \leq \frac{b_{N+1,1}-a_{i,1}}{N+1-i}$ for all $i \leq N$ (i.e. the arrival interval of aircraft $N+1$ does not decrease the overall minimum). Take $t_{N+1} = b_{N+1,1}$. Now call $b'_{N,1} = \min\{b_{N+1,1} - m(1,N), b_{N,1}\}$.

- If $b'_{N,1} = b_{N,1}$, we have successfully constructed $(t_1,\cdots,t_{N+1})$: t_{N+1} satisfies $t_{N+1} - t_N \geq m(1,N+1) = m(1,N)$, and for all $(i,j) \in \{1,\cdots,N\}^2$ such that $i<j$, $t_j - t_i \geq m(1,N) = m(1,N+1)$.
- If $b'_{N,1} = b_{N+1,1} - m(1,N)$. Let us rename every other $b_{i,1}$ as $b'_{i,1}$, for $i < N$, for notational ease. Now define $\underline{m} = \min_{i<N}\left(\frac{b'_{N,1}-a_{i,1}}{N-i}\right)$. Now

$$\underline{m} = \min_{i<N}\left(\frac{b'_{N,1}-a_{i,1}}{N-i}\right) = \min_{i<N}\left(\frac{b_{N+1,1}-m(1,N)-a_{i,1}}{N-i}\right) \tag{9}$$

  Thus

$$\underline{m} = \min_{i<N}\left(\frac{b_{N+1,1}-a_{i,1}}{N+1-i}\frac{N+1-i}{N-i} - \frac{m(1,N)}{N-i}\right) \geq m(1,N)\min_{i<N}\left(\frac{N+1-i}{N-i} - \frac{1}{N-i}\right) \tag{10}$$

  which proves $\underline{m} \geq m(1,N)$, and therefore the change in $b_{N,1}$ into $b'_{N,1} = b_{N+1,1} - m(1,N)$ does not decrease (8). We now have successfully constructed $(t_1,\cdots,t_{N+1})$: $t_{N+1} - t_N \geq m(1,N+1) = m(1,N)$, and for the N-tuplet $(t_1,\cdots,t_N)$, we just proved that $\underline{m} \geq m(1,N)$ is achievable.

Case 2: $m(1,N+1) = \min_{i<N+1}\left(\frac{b_{N+1,1}-a_{i,1}}{N+1-i}\right) < m(1,N)$. Obviously, adding a new aircraft has restricted the available spacing of the others: $\exists l \in \{1,\cdots,N\}$ such that $m(l,N+1) = m(1,N+1) < m(1,N)$.

- If $l=1$, $m(1,N+1) = \frac{b_{N+1,1}-a_{1,1}}{N}$. The evenly spaced solution is the optimal solution. Let us show it is feasible, i.e. $\forall j \in \{2,\cdots,N\}$, $a_{1,1} + \frac{j-1}{N}(b_{N+1,1} - a_{1,1}) \in [a_{j,1}, b_{j,1}]$. If this were not the case, then, without loss of generality[2], we would have $a_{1,1} + \frac{j_0-1}{N}(b_{N+1,1} - a_{1,1}) > b_1^{j_0}$ for a particular j_0, which would mean $\frac{b_{j_0,1}-a_{1,1}}{j_0-1} < \frac{b_{N+1,1}-a_{1,1}}{N}$, which is by assumption wrong.
- If $l \geq 2$, we can use the symmetry of the problem to flip it (the $a_{i,1}$ and $b_{i,1}$ are inverted)[3], and we are now in case 1.

In case 1 and case 2, we were able to construct a solution such that $\Delta_\pi = m(1,N)$ for every N. By Lemma 1, this is the best Δ_π, i.e. Δ^*, that we can achieve, and equation (6) follows. □

2.3 Transformation of the feasibility problem into a scheduling problem

The previous section solved for the maximal available spacing in the case of fixed order of arrival. In the current ATC system, if one interprets $a_{i,1}$ as the nominal time of arrival of aircraft i, and $(a_{i,1}, b_{i,1}]$ as the set of possible arrival times with delays, equation (6) predicts the best available spacing without altering the order of arrival (first come first served policy). Despite the fact that it is used almost all the time in

[2] Since the problem is symmetric.

[3] We treat first the subset $\{l\cdots N+1\}$ of aircraft (which we can do using the induction assumption, since $l>1$, and thus $(N+1-l<N)$). We then complete with the set $\{1\cdots,l-1\}$, but this time we will be in case 1 because the argmin of (6) is obtained in the set $\{l\cdots N+1\}$.

practice, this policy is of course not optimal.[4] We therefore now allow to change the order of arrival of the aircraft.

We show that the feasibility problem of (3) can be reduced to a single processor scheduling problem with release times and deadlines [17], known to be solvable in polynomial time [2, 10, 11, 9, 12]. The two following problems are equivalent:

Problem 1: Let Δ be a given positive number (separation). Given a set of N intervals $[a_{i,1}, b_{i,1}]$, $i \in \{1, \cdots N\}$, find a set $\{t_i\}_{i\in\{1,\cdots N\}}$ such that $\forall i \in \{1, \cdots N\}$, $t_i \in [a_{i,1}, b_{i,1}]$ and $\forall (i,j) \in \{1, \cdots, N\}^2$, s.t. $i > j$, $|t_i - t_j| \geq \Delta$.

Problem 2: Let D be a given positive number (processing time). Let $I = \{1, \cdots, N\}$ be a set of jobs. Let each job have a release time r_i and a deadline d_i. Find an ordering $\{t_i\}_{i\in I}$ such that $t_i \geq r_i$ (job i cannot start before the release time r_i), $t_i + D \leq d_i$, (job i has to be processed before the deadline d_i) $t_i < t_j \Longrightarrow t_j \geq t_i + D$ (two jobs cannot be processed simultaneously.

Solving problem 2 for a given set of $[r_i, d_i]$ and D, is thus equivalent to finding a feasible solution to our original problem, with $a_{i,1} = r_i$ and $b_{i,1} + D = d_i$. Problem 2 is a known problem in scheduling, for which polynomial algorithms have been developed [2, 10, 11, 9, 12]. We will use a slight modification of Carlier's algorithm [10] to solve the feasibility problem (Problem 2). This algorithm is presented in Appendix 1.

3 Algorithm

We call Carlier the algorithm presented in the Appendix. Carlier solves Problem 2 in section 2.3. We will use the following notation: Carlier(r_i, d_i, D) represents the ordering obtained with release times r_i, deadlines d_i and processing time D if such a schedule exists.

Theorem 2. *Assume that $n_i = 1$ for all $i \in \{1, \cdots N\}$, $a_{i,1} \in \mathbb{N}$, $b_{i,1} \in \mathbb{N}$, $D \in \mathbb{N}$. The following bisection algorithm converges to the exact solution of (3) in a finite number of steps. The time complexity of the algorithm is $O(N^2 \log(N)(N + \log L))$ where $L = \max b_{i,1} - \min a_{i,1}$.*

[4] Take for example $a_{1,1} = 00:00:00$, $a_{2,1} = 00:00:30$, $b_{1,1} = 00:03:00$, $b_{2,1} = 00:01:30$, $n_1 = n_2 = 1$.

1 $\Delta := \frac{1}{N-1}\left(\max_{i\in\{1,\cdots N\}} b_{i,n_i} - \min_{i\in\{1,\cdots N\}} a_{i,1}\right)$
2 **if** *Carlier*$(a_{i,1}, b_{i,1}+\Delta, \Delta)$ *is feasible,* **return** Δ
3 **while** *Carlier*$(a_{i,1}, b_{i,1}+\Delta, \Delta)$ *is not feasible,* $\Delta := \Delta/2$
4 $\underline{\Delta} := \Delta, \quad \overline{\Delta} := 2\cdot\Delta$
while $\left((\overline{\Delta} - \underline{\Delta}) \geq \frac{1}{N-1}\right)$
if *Carlier*$\left(a_{i,1}, b_{i,1} + \frac{\underline{\Delta}+\overline{\Delta}}{2}, \frac{\underline{\Delta}+\overline{\Delta}}{2}\right)$ *is feasible*
$\overline{\Delta} := \frac{\underline{\Delta}+\overline{\Delta}}{2}$
else
$\underline{\Delta} := \frac{\underline{\Delta}+\overline{\Delta}}{2}$
5 **return** $\max\left\{\Delta | \Delta \in [\underline{\Delta}, \overline{\Delta}] \cap \bigcup_{k=1}^{N-1}\left(\frac{\mathbb{N}}{k}\right) \wedge \textit{Carlier}(a_{i,1}, b_{i,1}+\Delta, \Delta)\ \textit{feasible}\right\}$

Proof — **Convergence** If $\Delta = \frac{1}{N-1}[b_{N,n_N} - a_{1,1}]$ is feasible, the algorithm stops at step 2. This is the ideal case (the largest achievable spacing is feasible).

Otherwise, let us call Δ^* the optimum of the problem. Let us call $\nu(\cdot)$ the order of arrival of the aircraft (aircraft i arrives in position $\nu(i)$). By Theorem 1, we have:

$$\Delta^* = \min_{(i,j)\in\{1,\cdots,N\}^2,\ i<j} \left(\frac{b_{\nu(j),1} - a_{\nu(i),1}}{\nu(j)-\nu(i)}\right) \tag{11}$$

Because of the limited possible values of the difference $\nu(j)-\nu(i)$, and because of the fact that the $a_{i,1}$, and $b_{i,1}$ are integers, equation (11) implies:

$$\Delta^* \in \bigcup_{k=1}^{N-1} \frac{\mathbb{N}}{k} \tag{12}$$

where $\bigcup_{k=1}^{N-1} \frac{\mathbb{N}}{k} = \{\frac{p}{q} \mid q \in \{1,\cdots N-1\},\ p \in \mathbb{N}\} \subset \mathbb{Q}$. In step 3, the algorithm will divide Δ by 2 until it becomes feasible. In step 4, starting from the last value of step 3, it will approach Δ^* by bisection until it comes within $\frac{1}{N-1}$ of Δ^*. Then we know that $\Delta^* \in [\underline{\Delta}, \overline{\Delta}]$. Because of the previous remark on Δ^*, $\Delta^* \in [\underline{\Delta}, \overline{\Delta}] \cap \bigcup_{k=1}^{N-1}\left(\frac{\mathbb{N}}{k}\right)$. But this set contains at most $N-1$ elements since $\overline{\Delta} - \underline{\Delta} \leq \frac{1}{N-1}$, which are easy to enumerate (step 5). Δ^* is the largest of them.

Complexity The number of calls to Carlier is 1 for step 2, and $N-1$ for step 5. The worst case for step 3 is if Δ has to be divided by 2 p times until

$$\frac{1}{N-1} \geq \frac{\max b_{i,1} - \min a_{i,1}}{N-1}\frac{1}{2^p}$$

and the same applies for step 4, which gives the following bound for the number of calls to Carlier: [5]

$$N + \left\lceil \frac{2}{\log 2} \log\left(\max b_{i,1} - \min a_{i,1}\right)\right\rceil \tag{13}$$

The number of calls of the procedure Carlier is thus $O(N+\log L)$, where $L = \max b_{i,1} - \min a_{i,1}$. The complexity of the modified Carlier's algorithm is $O(N^2\log(N))$ (see Appendix 1). The complexity of the algorithm is thus $O(N^2\log(N)(N+\log L))$. □

[5] Combining the length of the two intervals might give a slightly better bound than (13) but does not change the complexity.

Corollary 3. *Theorem 2 also holds for $a_{i,j} \in \mathbb{Q}$ and $b_{i,j} \in \mathbb{Q}$.*

Proof — Call $a_{i,j} = \frac{p_i^j}{q_i^j}$ and $b_{i,j} = \frac{r_i^j}{s_i^j}$, and call L the least common denominator of the q_i^j and s_i^j (note that $L = \prod_{i=1}^{N} \prod_{j=1}^{n_i} q_i^j s_i^j$ works as well). Then problem (2) is equivalent to the following problem:

$$\begin{array}{ll} \textbf{Maximize:} & L\delta \\ \textbf{Subject to:} & L\theta_i \in \bigcup_{j=1}^{n_i} [L\frac{p_i^j}{q_i^j}, L\frac{r_i^j}{s_i^j}] \\ & |L\theta_i - L\theta_j| \geq L\delta \qquad \forall (i,j) \in \{1, \cdots, N\}^2, \text{ s.t. } i > j \end{array} \tag{14}$$

In the previous formula, $L\frac{p_i^j}{q_i^j} \in \mathbb{N}$ and $L\frac{r_i^j}{s_i^j} \in \mathbb{N}$, so Theorem 2 gives us a solution $(\Delta^*, t_1, \cdots, t_N) = (L\delta^*, L\theta_1, \cdots, L\theta_N)$. Therefore, $(\delta^*, \theta_1, \cdots, \theta_N)$ solves the problem in $\mathbb{Q}$. □

4 Numerical performance of the algorithm

The running time of the algorithm is theoretically bounded above by $O(N^2 \log(N)(N + \log L))$, where $O(\cdot)$ means there is a constant in front of $N^2 \log(N)(N + \log L)$ and it is independent of N and L. To verify this time bound we have simulated a set of 1800 cases and compare the measured CPU time used by our algorithm and that using CPLEX to solve the same problem, to demonstrate the efficiency of our algorithm ($N \in \{2, \cdots, 19\}$). We have run 8100 additional simulations ($N \in \{20, \cdots, 100\}$) to assess the performance of our algorithm in a range of N where CPLEX cannot handle the computation in real time.

We simulate situations in which N aircraft are within one hour of the destination airport. For each value of N, we randomly generate intervals $[a_{i,1}, b_{i,1}]$ within one hour of Oakland, with various width $b_{i,1} - a_{i,1}$ ranging from 30 sec. (almost no possibility to adjust the arrival time of the aircraft) to 15 min. We measure the CPU time needed by our method to compute the largest available Δ^* between the aircraft. We run 100 simulations for each N.

Our algorithm is implemented in MATLAB; the CPLEX solution is coded in AMPL interfaced with MATLAB, so that both methods can run from the same application on the same platform.[6] The results of these runs are shown in Figure 4. This figure suggests a few comments. The average result is not representative of the relative performance of the two algorithms. Looking at Figure 4 (left) would suggest that CPLEX's performance is much worse than our algorithm for $N \geq 14$. Figure 4 (right) shows that the average is "polluted" by abnormal runs: in fact for the set of simulations shown here, CPLEX is faster than our algorithm in 85% of the cases, but among the 15% remaining, the CPU time used by CPLEX

[6] We use a SOLARIS 8 workstation under UNIX (2GB of RAM). This is not important as long as the two methods run on the same machine. Comment: The MATLAB implementation is quite unfavorable to our algorithm. MATLAB is notoriously slow for conditional operation such as "if", "else"... Note that the CPLEX/AMPL solution procedure is coded separately, and MATLAB is used for it only a model interface.

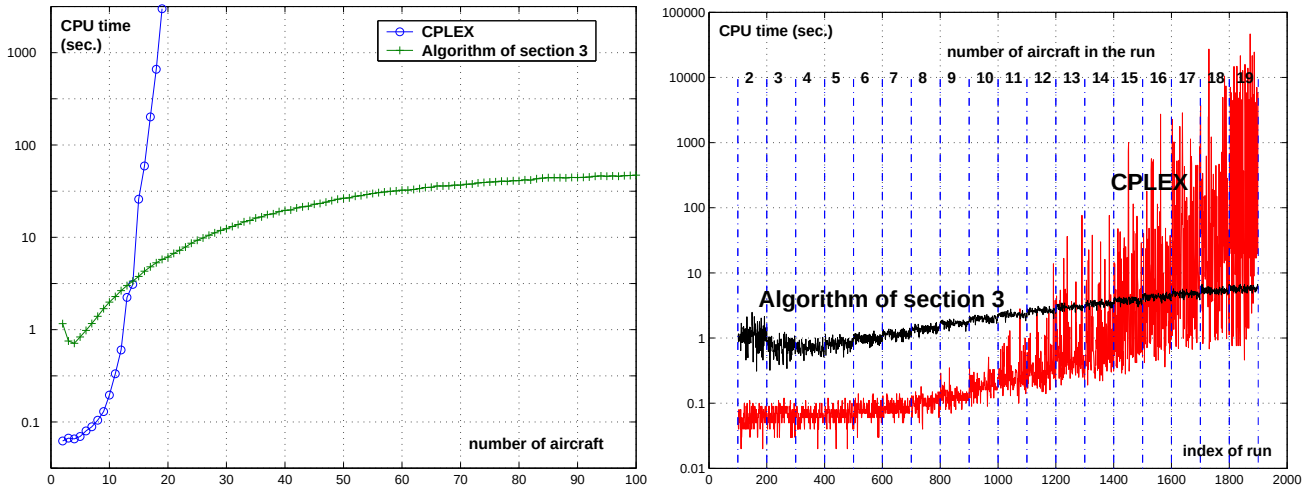

Figure 4: **Left:** CPU time used by our algorithm and by CPLEX to solve the same problems. The curve is an average over 100 runs for each value of N (1800 runs for CPLEX, 9900 runs for our algorithm). **Right:** Comparison of the results for the first 1800 simulations realized. As can be seen, even for $N \geq 14$, a significant number of CPLEX computations are faster than our algorithm by at least one order of magnitude.

exceeds the CPU time used by our algorithm by several orders of magnitude, which increases the average significantly.

We are also aware that there might be more efficient ways to encode the MILP (2) in CPLEX or even the one interval version (3). For example, an other way to encode (2) is:

$$\begin{array}{lll} \textbf{Maximize:} & \Delta & \\ \textbf{Subject to:} & t_i \geq a_{i,n_i} + \sum_{j=1}^{n_i-1} x_{ij}(a_{i,j} - a_{i,j+1}) & \forall i \in \{1, \cdots N\} \\ & t_i \leq b_{i,n_i} + \sum_{j=1}^{n_i-1} x_{ij}(b_{i,j} - b_{i,j+1}) & \forall i \in \{1, \cdots N\} \\ & x_{ij} \leq x_{i\ j+1} & \forall i \in \{1, \cdots N\}\ \forall j \in \{1, \cdots n_i - 2\} \\ & x_{ij} \in \{0, 1\} & \forall i \in \{1, \cdots N\}\ \forall j \in \{1, \cdots n_i - 1\} \end{array} \quad (15)$$

This formulation (suggested to us by Francis Carr) has been observed by other researchers to be more efficient than (3). Regardless of the respective benefits of different MILP formulations of the original problem (1), the goal of our algorithm is to provide a guarantee on running time for a given task, which CPLEX does not provide, as is illustrated by Figure 4.

5 Current work and open problems

Section 5.1 shows how to adapt the algorithm of section 3 for the case in which $n_i > 1$, but the order of the aircraft is known a priori. It summarizes the computational complexity for the three cases which we were able to solve. Section 5.2 suggests some simplifications of the problem, relevant to Air Traffic Control, which are currently the focus of our research.

5.1 Contributions of this article

Unfortunately, the transformation operated in the one interval case, to cast our problem into a scheduling problem form, does not work in the multi-interval case: for aircraft i, call $[a_{i,1}, b_{i,1}]$ and $[a_{i,2}, b_{i,2}]$ the two first feasible arrival time intervals. Let us suppose that $a_{i,2} - b_{i,1} < \Delta$. Applying the previous transformation to the set of two intervals would result in a single interval $[a_{i,1}, b_{i,2} + \Delta]$. Scheduling an aircraft (or equivalently a job) at time $\frac{1}{2}(b_{i,1} + a_{i,2})$ does not represent a physically acceptable solution, because it is in none of the arrival time intervals of the aircraft. However, nothing would prevent an algorithm from doing it. Even if it did, Carlier's algorithm does not generalize easily to multiple intervals.

For the $n_i > 1$ case with fixed order of arrival, we replace the call of the modified Carlier's algorithm by the following procedure construct(Δ) (in which we assume without loss of generality that the aircraft are sorted by ascending $a_{i,1}$, i.e. $a_{1,1} \leq a_{2,1} \leq \cdots \leq a_{N,1}$):

---

1 $t_1 := a_{1,1}$
2 **for** $i = 2 : N$
3 **if** $\left\{ t \mid t \in [t_{i-1} + \Delta, b_{i,n_i}] \cap \left\{ \bigcup_{j=1}^{n_i} [a_{i,j}, b_{i,j}] \right\} \right\} = \emptyset$
return false
else
3' $t_i = \min \left\{ t \mid t \in [t_{i-1} + \Delta, b_{i,n_i}] \cap \left\{ \bigcup_{j=1}^{n_i} [a_{i,j}, b_{i,j}] \right\} \right\}$
end
end

If this procedure does not return false, the t_i are a feasible schedule which satisfies $\forall i > j,\ t_i > t_j + \Delta$. It is quite easy to see that the same proof as in section 3 applies, and that the algorithm also stops in a finite number of steps. Calling $M = \sum_{i=1}^{N} n_i$, one can easily show that the complexity of the algorithm is $O(M(N + \log L))$. The contributions of this paper are thus three algorithms, whose complexity is summarized in the following array:

	order fixed	order not fixed
$n_i = 1$	$O(N^2)$ closed form	$O\left((N^2 \log N)(N + \log L)\right)$ exact
$n_i > 1$	$O(M(N + \log L))$ exact	? ?

N is the number of aircraft, $M = \sum_{i=1}^{N} n_i$, $L = \frac{1}{N}\left(\max_{i=1}^{n_i} b_{i,n_i} - \min_{i=1}^{n_i} a_{i,1}\right)$.

5.2 Open problems and conclusion

Besides the bottom right cell of the array above, several other questions raised by this problem remain open. If we transform the problem and assume $S_i = [a_{i,1}, b_{i,1}] + \mathbb{N}T$ (which physically corresponds to an aircraft which could execute a large number of holding patterns before landing), we still do not know if finding the minimum makespan (landing time of the last aircraft to land) is polynomial solvable. Furthermore, minimizing the total number of holding patterns, minimizing the total number of aircraft put on a holding patterns, minimizing the sum of all times of arrival and minimizing the total makespan are not equivalent problems. Our current research efforts are aimed at deriving approximation algorithms for these four problems, i.e. polynomial time algorithms, for which we can guarantee that they converge to a value within a certain bound of the mathematical optimum.

Acknowledgments

We are grateful to Tom Schouwenaars for his help on CPLEX/AMPL, to Francis Carr, for his interest in this problem, as well as his suggestions on the MILP formulation of the problem, to Gano Chatterji for the suggestions which went into modeling this problem, and to George Meyer for his support in this project.

References

[1] R. K. Ahuja, T .L. Magnanti, and J. B. Orlin. *Network Flows, Theory, Algorithms and Applications.* Prentice Hall, Upper Saddle River, NJ, 1993.

[2] P. Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine when processing times are equal. *Journal of Scheduling*, 2:245–252, 1999.

[3] A. M. Bayen, P. Grieder, C. J. Tomlin, and G. Meyer. Control theoretic Lagrangian models for sector-based air traffic flow. Submitted Dec. 2002 to the *AIAA Journal on Guidance, Dynamics and Control*, 2002.

[4] A. M. Bayen and C. J. Tomlin. Real-time discrete control law synthesis for hybrid systems using MILP: applications to congested airspaces. To appear in the *Proceedings of the 2003 American Control Conference.*

[5] A. Bemporad, F. Borrelli, and M. Morari. Piecewise linear optimal controllers for hybrid systems. In *2000 American Control Conference*, pages 1190–1194, Chicago, IL, June 2000.

[6] A. Bemporad, F. Borrelli, and M. Morari. On the optimal control law for linear discrete time hybrid systems. In C.J. Tomlin and M. Greenstreet, editors, *Hybrid Systems: Computation and Control*, LNCS 2289, pages 105–119. Springer Verlag, 2002.

[7] D. Bertsimas and J. N. Tsitsiklis. *Introduction to linear optimization.* Athena Scientific, Belmont, MA, 1997.

[8] K. Bilimoria, B. Sridhar, G. Chatterji, K. Seth, and S.Grabbe. FACET: Future ATM concepts evaluation tool. In *3rd USA/Europe Air Traffic Management R&D Seminar*, Naples, Italy, June 2001.

[9] J. Carlier. Problème à une machine, rep. no. 78.05. Technical report, Institut de Programmation, Université Pierre et Marie Curie, Paris, France, 1978.

[10] J. Carlier. Problèmes d'ordonnancement à durées égales. *QUESTIO*, 5(4):219–229, 1981.

[11] J. Carlier. The one machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.

[12] J. Carlier. *Problèmes d'ordonnancement à contraintes de ressources: algorithmes et complexité.* PhD thesis, Université Pierre et Marie Curie (Paris VI), 1984.

[13] D. Dugail, E. Feron, and K. Bilimoria. Conflict-free conformance to en route flow-rate constraints. In *AIAA Conference on Guidance, Navigation and Control*, Monterey, CA, August 2002.

[14] H. Erzberger, H. T. Davis, and S. M. Green. Design of Center-TRACON Automation System. In *AGARD Meeting on Machine Intelligence in Air Traffic Management*, Berlin, Germany, May 1993.

[15] C. A. Floudas. *Nonlinear and mixed-integer programming - fundamentals and applications.* Oxford University Press, Oxford, UK, 1995.

[16] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: a modeling language for mathematical programming.* Boyd and Fraser, Danvers, MA, 1999.

[17] M. R. Garey and D. S. Johnson. *Computers and intractability, a guide to the theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[18] J. Histon and R. J. Hansman. The impact of structure on cognitive complexity in air traffic control. Technical Report ICAT-2002-4, MIT, June 2002.

[19] D. Jackson and J. Chapin. Redesigning air traffic control: An exercise in software design. *IEEE Software*, 17(3):63–70, 2000.

[20] V. Jain and I. E. Grossmann. Algorithms for hybrid MILP / CP models for a class of optimization problems. *INFORMS Journal on Computing*, 13:258–276, 2001.

[21] A. Richards, J. How, and E. Feron. Plume avoidance maneuver planning using mixed integer linear programming. In *AIAA Conference on Guidance, Navigation and Control*, Montreal, Canada, August 2001.

[22] A. Richards, T. Schouwenaars, J. How, and E. Feron. Spacecraft trajectory planning with collision and plume avoidance using mixed-integer linear programming. *AIAA Journal of Guidance, Control and Dynamics*, 25(4):755–764, 2002.

[23] V. V. Vazirani. *Approximation Algorithms.* Springer Verlag, Berlin, Germany, 2001.

[24] H. P. Williams and S. C. Brailsford. Computational logic and integer programming. In J. Beasley, editor, *Advances in Linear and Integer Programming*, pages 249–281. Oxford University Press, 1996.

Appendix: Modified Carlier's algorithm

This Appendix presents a modified version of Carlier's algorithm [10, 12] for solving the feasibility of problem (3) for a given Δ, when $n_i = 1$. We were not able to reprove Carlier's original algorithm (we think that the proposed example in the paper has a minor error), thus, the algorithm presented here includes a modification by us. We present this complete algorithm and our proof (which is a slightly modified version of Carlier's) here, as the original algorithm is published in French and may not be accessible to some.

We reproduce the work of Carlier [10, 12] as much as possible, using the original notation of the author, and modifying it when necessary. The problem solved by Carlier is the following.

Let I be a set of jobs. We want to find an ordering[7] *$\mathcal{T} = (t_i)_{i \in I}$ such that*

1. *$t_i \geq a_i$, job i cannot start before time a_i.*
2. *$t_i + D \leq b_i$, job i cannot end after time b_i.*
3. *$t_i < t_j \Longrightarrow t_j \geq t_i + D$: two jobs cannot be executed simultaneously.*

A few definitions and some notation need to be given and will be used in the algorithm below. The algorithm recursively builds an ordering $\mathcal{T}^x$ satisfying the conditions above, where the recursion is run on x which represents time (therefore, x will be proceed from $\min_i a_i$ to $\max_j b_j$).

- $K_x = \{i | a_i + D \leq x\}$ is the set of jobs which can be finished before time x.
- Let $\mathcal{T}^x$ be an ordering of the jobs constructed by the algorithm; we call U_x the set of jobs ordered by $\mathcal{T}^x$: $U_x = \{i | i \text{ ordered by } \mathcal{T}^x\}$.
- $H = K_x - U_{x-D}$ is the set of jobs which can be finished before time x (i.e. from K_x), which have not been ordered by $\mathcal{T}^{x-D}$ (i.e. not in U_{x-D}).
- m is the most urgent job of H: $m = \arg\min_{h \in H} b_h$.
- The delay of an ordering $\mathcal{T}$ (with corresponding set of indices U) is: $y = \max_{u \in U}(t_u + D)$. The delay of $\mathcal{T}$ is the time this ordering takes to complete.
- An ordering $\mathcal{T}$ (with corresponding set of indices U) *active* as follows: an ordering $\mathcal{T}$ (with corresponding set of indices U) is said to be *active* iff there does not exist a job w in the complement $\overline{U}$ of U such that $\max_{u \in U}(t_u + D) > b_w - D$. In other words, there does not exist a job w in the complement of U which cannot be processed before its deadline b_w because w cannot start before the completion time (delay) of the jobs in U. Mathematically, it equivalently reads: $\max_{u \in U}(t_u + D) \leq \min_{u \in \overline{U}} b_w - D$.

[7] In French: "ordonnancement"

- An ordering $\mathcal{T}$ (with corresponding set of indices U) is said to be *x-active* iff it is active and has a delay less than x.
- $\{\mathcal{T}^{x-D}+m\}$ is the ordering obtained by adding the most urgent job m to the ordering $\mathcal{T}^{x-D}$, if m is executed as soon as possible. As soon as possible means the job m should start at $t_m = \max\{a_m, y\}$, where $y = \max_{u \text{ in } \mathcal{T}^{x-D}}(t_u + D)$ is the delay of the ordering $\mathcal{T}^{x-D}$.
- $T(x)$ is the set of orderings which are active and which have a delay less than x.

Definition 1. (Carlier) *The operator "preferable to" $\succeq$ is defined the following way: for two sets U and V of elements of I, $U \succeq V$ if:*

1. $p = |U| \geq r = |V|$
2. $b_{u_1} \leq b_{v_1}, \cdots, b_{u_r} \leq b_{v_r}$ *if* $b_{u_1} \leq b_{u_2} \cdots \leq b_{u_p}$ *and* $b_{v_1} \leq b_{v_2} \cdots \leq b_{v_r}$.

Theorem 4 (after Carlier [10, 12]). *$\forall x \in \{\min_{i\in I} a_i, \cdots, \max_{i\in I} b_i\}$, the set $\mathcal{T}^x$ constructed by the algorithm below is x-active and is preferable to any other x-active ordering.*

```
1  ∀x ∈ {min_{i∈I} a_i, ··· , min_{i∈I} a_i + D − 1}, T^x := ∅, U_x := ∅
2  for x = min_{i∈I} a_i + D : max_{i∈I} b_i
3     K_x := {i | a_i + D ≤ x}, H = K_x − U_{x−D}
      if H = ∅
4        T^x = T^{x−D}
      else
5        m = arg min_{h∈H} b_h
         if {T^{x−D} + m} is active
6           T^x = {T^{x−D} + m}
         else
7           T^x = T^{x−1}
         end
      end
   end
8  if U_x ≠ I no solution else return T^x end
```

Proof — The proof is done by induction on x. Obviously, $\forall x \in \{\min_{i\in I} a_i, \cdots, \min_{i\in I} a_i + D - 1\}$, the property is true, since $\mathcal{T}^x := \emptyset$ and $U_x := \emptyset$.

Suppose the property is true for all $z \leq x-1$. We want to show that it is true for $\mathcal{T}^x$. There are two cases:

<u>**Case 1:**</u> $H = \emptyset$.

We know that $K_x \subseteq U_{x-D}$ because $K_x - U_{x-D} = \emptyset$. Since the set of jobs ordered at time $x - D$ is at most the number of jobs that *can* be ordered at time $x - D$, $U_{x-D} \subseteq K_{x-D} \subseteq K_x$, because K_x is monotonically increasing. Combining the above inclusions, we have $K_x = U_{x-D}$. This means that all jobs executable before time x are ordered by U_{x-D} (i.e. by $\mathcal{T}^{x-D}$). Also, $\mathcal{T}^{x-D} \in T(x-D) \subseteq T(x)$, therefore $\mathcal{T}^{x-D} \in T(x)$. This implies that $\mathcal{T}^{x-D}$ is a maximal element of $T(x)$ for the preorder $\succeq$.

Case 2: $H \neq \emptyset$.

H contains at least one element which can be finished before x not ordered by U_{x-D}. Let $\mathcal{T}' = \{\mathcal{T}^{x-D} + m\}$; let $\mathcal{T}$ be any ordering in $T(x)$, and call l the last job which was ordered by $\mathcal{T}$, and $\mathcal{T} - \{l\}$ the ordering obtained by withdrawing job l from $\mathcal{T}$, U the set of jobs ordered by $\mathcal{T}$.

- We first show that if $\mathcal{T}'$ is active, then $\mathcal{T}' \succeq \mathcal{T}$. We use Carlier's [10, 12] lemma on the preorder $\succeq$:
  Lemma: (Carlier [10, 12]) Let C and E be two sets such that $C \succeq E$. Let $u \notin C$ and $v \notin E$. Suppose that $\forall w \in \overline{C}$, $b_u \leq b_w$. Then $\{u\} \cup C \succeq \{v\} \cup E$. $\square$.
  We can apply the previous lemma to the present problem: take $I = K_x$, $C = U_{x-D}$, $E = U - \{l\}$, $u = m$, $v = l$. In the following, the complement of a set $K \subseteq I$ is defined as $\overline{K} = I \backslash K$. Clearly, $u \notin C$, $v \notin E$, $\forall w \in \overline{C}$, $b_u \leq b_w$ because by construction, $m = u$ is the most urgent job. Finally $U_{x-D} \succeq U - \{l\}$ by the induction assumption. Therefore $\mathcal{T}' \succeq \mathcal{T}$, that is, $\mathcal{T}'$ is preferable to any element of $T(x)$. Since $\mathcal{T}'$ is active, it must also be x-active. To see this, we need to prove that the delay of $\mathcal{T}'$ is less than x. Calling y the delay of $\mathcal{T}^{x-D}$, we have $y \leq x - D$. Since $y \leq \min_{w \in \overline{\mathcal{T}^{x-D}}} b_w - D$, because $\mathcal{T}^{x-D}$ is $x - D$ active and $m \in \overline{\mathcal{T}^{x-D}} \cap K_x$, m can be executed before x, i.e. $t_m + D \leq x$. In conclusion, $\mathcal{T}'$ is x-active and preferable to all elements of $T(x)$.
- Second, we show that if $\mathcal{T}'$ is not active, then $\mathcal{T}^x = \mathcal{T}^{x-1}$. We want to show that $\forall \mathcal{T} \in T(x)$, $\mathcal{T}^{x-1} \succeq \mathcal{T}$.
  - If $\max_{u \in \mathcal{T}}(t_u + D) \leq x - 1$, this result is immediate by induction.
  - Otherwise $\max_{u \in \mathcal{T}}(t_u + D) = x$. We call l the last job ordered by $\mathcal{T}$ (see Figure 5). By induction, we know that $\mathcal{T}^{x-D} \succeq \mathcal{T} - \{l\}$. This implies [8] $\overline{\mathcal{T} - \{l\}} \succeq \overline{\mathcal{T}^{x-D}}$.
    Because $\{\mathcal{T}^{x-D} + m\}$ is not active, $\exists \overline{w} \in \overline{\{\mathcal{T}^{x-D} + m\}}$ such that $b_{\overline{w}} - D < x$. We have $\overline{w} \in \mathcal{T}$ since otherwise $\overline{w}$ can only start after x (the delay of $\mathcal{T}$) or $b_{\overline{w}} - D \geq x$, which is a contradiction. Thus, $\overline{w}$ must have been processed before x and $\overline{w} \in K_x$. Since m was the most urgent task added to $\mathcal{T}^{x-D}$, $b_m \leq b_{\overline{w}} < x + D$ (that is, both m and $\overline{w}$ are in K_x when m is added).
    Now we have: $m \in \overline{\mathcal{T}^{x-D}}$ and $\overline{w} \in \overline{\mathcal{T}^{x-D}}$. Since $\overline{\mathcal{T} - \{l\}}\ P\ \overline{\mathcal{T}^{x-D}}$, $\overline{\mathcal{T} - \{l\}}$ contains at least two jobs i and j, indexed by the order $b_i \leq b_j$, such that $b_i \leq b_m < x + D$ and $b_i \leq b_{\overline{w}} < x + D$. Furthermore, since both i and j are in $\overline{\mathcal{T} - \{l\}}$, one of them must be scheduled after l, say job j, with completion time at least $x + D$. This implies $b_j \geq x + D$, which contradicts the previous inequality. Therefore there cannot be $\mathcal{T} \in T(x)$ of exact delay x.

We have proved that regardless of how $\mathcal{T}^x$ was constructed, it is always x-active, and orders more jobs than any other x-active ordering $\mathcal{T}$. If a solution to the original problem exists (i.e. the problem is feasible), we know that the algorithm will find it, because for any x, it finds one x-active ordering with the maximal number of jobs. It suffices to march x until $\max_{i \in I} b_i$. $\square$

Theorem 5. *1. The computational complexity of calculating $\mathcal{T}^x$ at each step is $O(N)$. 2. It is possible to modify the algorithm such that the overall computational complexity is $O(N^3)$.*

[8] Carlier also shows $A \succeq B \Rightarrow \overline{B} \succeq \overline{A}$.

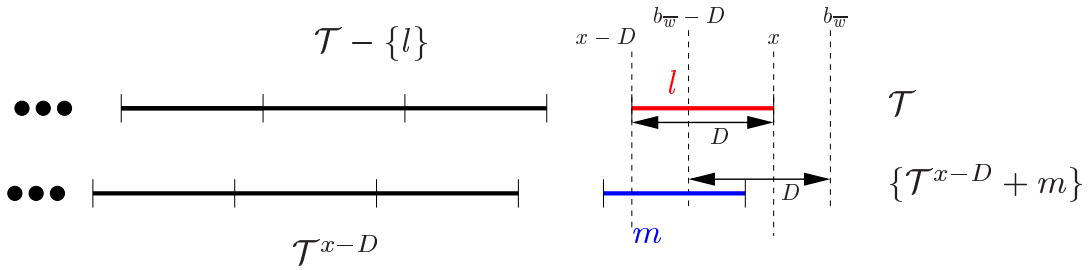

Figure 5: Case 2 from the proof of Theorem 4, with $\{\mathcal{T}^{x-D} + m\}$ not active. Orderings $\mathcal{T}$ of delay x exactly and $\{\mathcal{T}^{x-D} + m\}$ at iteration x. The last job l of $\mathcal{T}$ is such that $t_l + D = x$.

Proof —

1. In step 3, the computational complexity of calculating K_x, for all x from $\min_{i \in I} a_i + D$ to $\max_{i \in I} b_i$ is $O(N)$ (since it requires stepping through all a_i). Therefore, in the **for** loop of step 2, it is $O(1)$. The computation of H is at most $O(N)$, if the two sets K_x and U_{x-D} are constructed with increasing indices (i.e. the sets are indexed such that performing $K_x - U_{x-D}$ only requires checking the upper indices of K_x and U_{x-D}). All other operations are $O(1)$ except computing $m = \arg\min_{h \in H} b_h$. The cost of computing $\mathcal{T}^x$ is thus $O(N)$.
2. It suffices to compute the $\mathcal{T}^x$ for $x = a_i + kD$, with $i \in I = \{1, \ldots, N\}$ and $k \in I = \{0, \ldots, N-1\}$. The total number of x is N^2, which gives the algorithm a complexity of $O(N^3)$.

□

Remark —

- An upper bound on the number of $\mathcal{T}^x$ is $\max_{i \in I} b_i - \min_{i \in I} a_i$.
- The complexity can be reduced to $O(N^2 \min(N, D))$ with an additional test on U_x. At each iteration, we add the test $U_x = K_x$. If this is true, $x := \min_{i \in I - K_x} a_i$. The number of such jumps is at most N. Between two jumps, a number ν_i of jobs is scheduled, for a duration D each. $\sum_{i=1}^{N} \nu_i = N$, thus the number of x to step through for these jobs is at most ND. The total number of x to step through is thus $N + ND$. The complexity of the algorithm is thus $O(N^2 \min\{N, D\})$. In another article [11], Carlier shows that it is even possible to reduce the complexity of the algorithm to $O(N \log(N) \min(N, D))$.

□

We run Carlier's numerical example on the present algorithm, as an illustration of the algorithm. Consider six jobs, with following release times: $a_1 = 0$, $a_2 = 2$, $a_3 = 7$, $a_4 = 9$, $a_5 = 10$, $a_6 = 24$, and the following deadlines: $b_1 = 32$, $b_2 = 35$, $b_3 = 22$, $b_4 = 20$, $b_5 = 23$, $b_6 = 30$. The processing time of each job is $D = 5$. The following array now shows the results obtained on this example, with the algorithm above. For $x = 19$, $\{\mathcal{T}_{14} + 3\} = \{1, 4, 3\}$ is not active, according to the definition of active: the delay of this ordering is 19. This does not enable the later scheduling of job number 5: $b_5 = 23 < 19 + D = 24$. Therefore, $\mathcal{T}_{19} = \mathcal{T}_{18}$. Similarly, for $x = 20$, $U_{15} = \{1, 2, 4\}$, $m = 3$, but $\{1, 2, 4 + 3\}$ is not active, because its delay is 20, and $b_5 = 23 < 20 + D = 25$. $\{\mathcal{T}_{15} + m\}$ is not active, therefore $\mathcal{T}_{20} = \mathcal{T}_{19}$. The same is true for $\mathcal{T}_{21}$. For $x = 22$,

it becomes active again, and the jobs can be added properly until a full ordering is produced for $x = 32$ (see Figures 6 and 7).

x	K	H	m	$\{\mathcal{T}_{x-D} + m\}$ active	U_x	$\mathcal{T}_x$
5	$\{1\}$	$\{1\}$	1	yes	$\{1\}$	$\{0\}$
6	$\{1\}$	$\{1\}$	1	yes	$\{1\}$	$\{0\}$
7	$\{1, 2\}$	$\{1, 2\}$	1	yes	$\{1\}$	$\{0\}$
8	$\{1, 2\}$	$\{1, 2\}$	1	yes	$\{1\}$	$\{0\}$
9	$\{1, 2\}$	$\{1, 2\}$	1	yes	$\{1\}$	$\{0\}$
10	$\{1, 2\}$	$\{2\}$	2	yes	$\{1, 2\}$	$\{0, 5\}$
11	$\{1, 2\}$	$\{2\}$	2	yes	$\{1, 2\}$	$\{0, 5\}$
12	$\{1, 2, 3\}$	$2, 3$	3	yes	$\{1, 3\}$	$\{0, 7\}$
13	$\{1, 2, 3\}$	$2, 3$	3	yes	$\{1, 3\}$	$\{0, 7\}$
14	$\{1, 2, 3, 4\}$	$\{2, 3, 4\}$	4	yes	$\{1, 4\}$	$\{0, 9\}$
15	$\{1, 2, 3, 4, 5\}$	$\{3, 4, 5\}$	4	yes	$\{1, 2, 4\}$	$\{0, 5, 10\}$
16	$\{1, 2, 3, 4, 5\}$	$\{3, 4, 5\}$	4	yes	$\{1, 2, 4\}$	$\{0, 5, 10\}$
17	$\{1, 2, 3, 4, 5\}$	$\{2, 4, 5\}$	4	yes	$\{1, 3, 4\}$	$\{0, 7, 12\}$
18	$\{1, 2, 3, 4, 5\}$	$\{2, 4, 5\}$	4	yes	$\{1, 3, 4\}$	$\{0, 7, 12\}$
19	$\{1, 2, 3, 4, 5\}$	$\{2, 3, 5\}$	3	no	$\{1, 3, 4\}$	$\{0, 7, 12\}$
20	$\{1, 2, 3, 4, 5\}$	$\{3, 5\}$	3	no	$\{1, 3, 4\}$	$\{0, 7, 12\}$
21	$\{1, 2, 3, 4, 5\}$	$\{3, 5\}$	3	no	$\{1, 3, 4\}$	$\{0, 7, 12\}$
22	$\{1, 2, 3, 4, 5\}$	$\{2, 5\}$	5	yes	$\{1, 3, 4, 5\}$	$\{0, 7, 12, 17\}$
23	$\{1, 2, 3, 4, 5\}$	$\{2, 5\}$	5	yes	$\{1, 3, 4, 5\}$	$\{0, 7, 12, 17\}$
24	$\{1, 2, 3, 4, 5\}$	$\{2, 5\}$	5	yes	$\{1, 3, 4, 5\}$	$\{0, 7, 12, 17\}$
25	$\{1, 2, 3, 4, 5\}$	$\{2, 5\}$	5	yes	$\{1, 3, 4, 5\}$	$\{0, 7, 12, 17\}$
26	$\{1, 2, 3, 4, 5\}$	$\{2, 5\}$	5	yes	$\{1, 3, 4, 5\}$	$\{0, 7, 12, 17\}$
27	$\{1, 2, 3, 4, 5\}$	$\{2\}$	2	yes	$\{1, 3, 4, 5, 2\}$	$\{0, 7, 12, 17, 22\}$
28	$\{1, 2, 3, 4, 5\}$	$\{2\}$	2	yes	$\{1, 3, 4, 5, 2\}$	$\{0, 7, 12, 17, 22\}$
29	$\{1, 2, 3, 4, 5, 6\}$	$\{2, 6\}$	2	yes	$\{1, 3, 4, 5, 2\}$	$\{0, 7, 12, 17, 22\}$
30	$\{1, 2, 3, 4, 5, 6\}$	$\{2, 6\}$	2	yes	$\{1, 3, 4, 5, 2\}$	$\{0, 7, 12, 17, 22\}$
31	$\{1, 2, 3, 4, 5, 6\}$	$\{2, 6\}$	2	yes	$\{1, 3, 4, 5, 2\}$	$\{0, 7, 12, 17, 22\}$
32	$\{1, 2, 3, 4, 5, 6\}$	$\{6\}$	6	yes	$\{1, 3, 4, 5, 2, 6\}$	$\{0, 7, 12, 17, 22, 27\}$

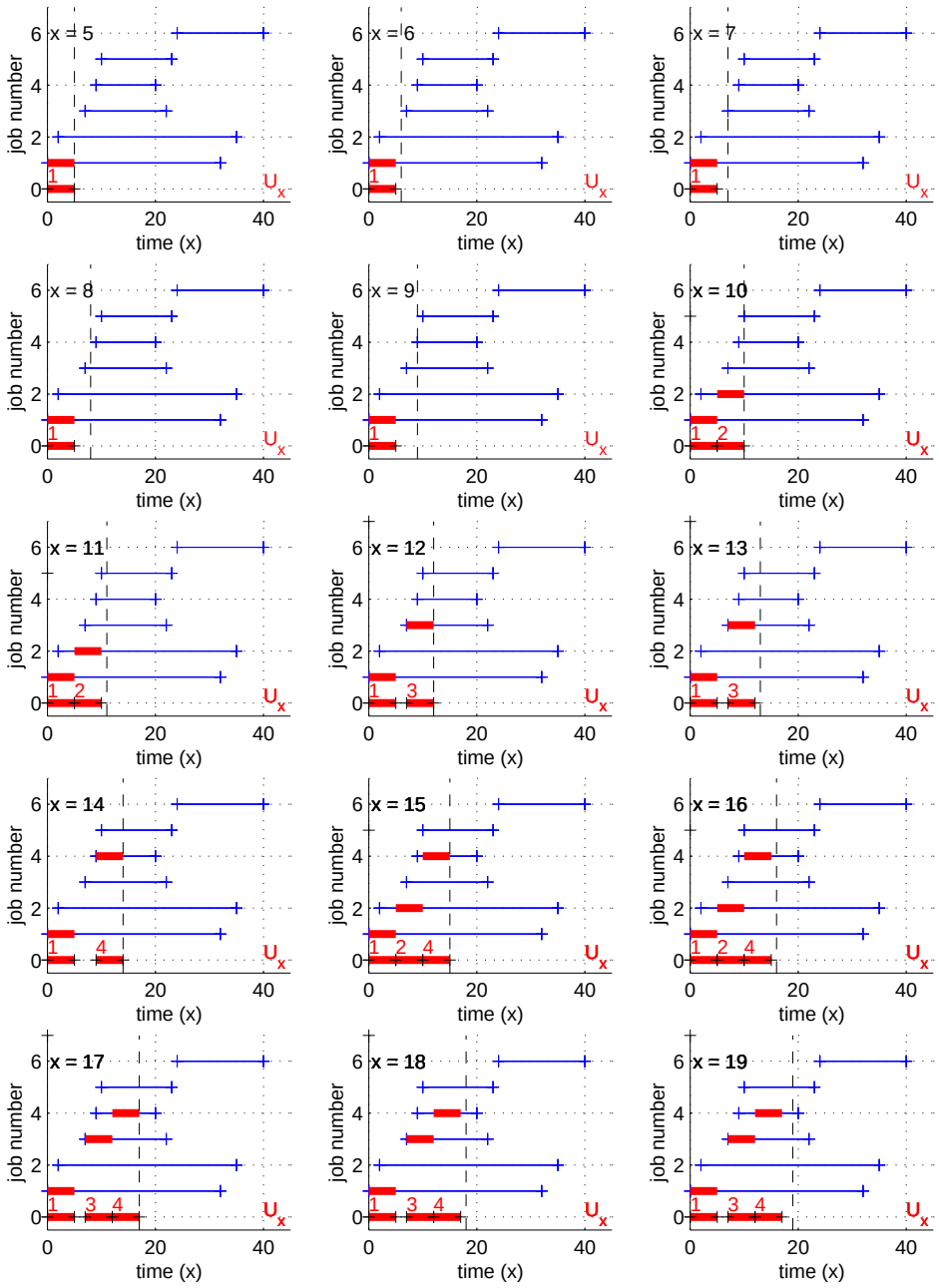

Figure 6: Successive orderings obtained with the modified Carlier algorithm for x in $\{5, \cdots, 19\}$. A strip on row i means that job i has been scheduled at the time corresponding to the beginning of the strip. Row 0 shows the set of all ordered jobs. At $x = 19$, $\{\mathcal{T}^{14} + m\}$ is not active according to our new definition, therefore, $U_{19} = U_{18}$.

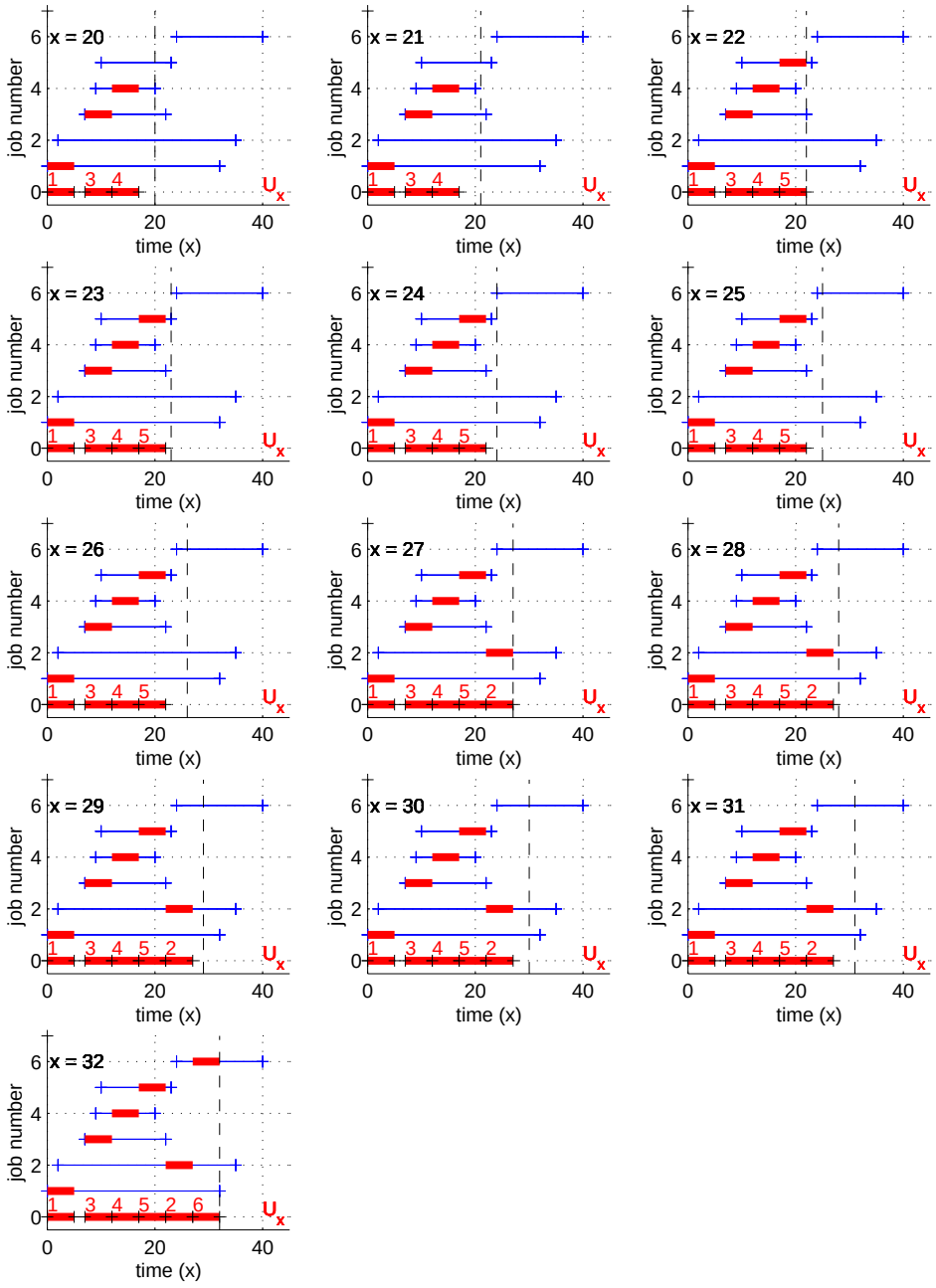

Figure 7: Successive orderings obtained with the modified Carlier algorithm for x in $\{20, \cdots, 32\}$. The same as in Figure 6 happens for $x = 20$ and $x = 21$. This prevents job 5 to be added in violation of its deadline ($x = 22$).