An Approximation Algorithm for Scheduling Aircraft with Holding Time†

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

Abstract—We consider the problem of scheduling arrival air traffic in the vicinity of large airports. The problem is posed as a single queue problem, from which aircraft can be pulled out and put “on hold”, in holding loops, each loop taking a fixed amount of time to traverse, before they join the queue again. The difficulty of deriving efficient solutions to this problem (which is currently controlled non optimally by human Air Traffic Controllers) is the minimization of “idle time” generated by traversing an integer number of loops. We formulate this problem as a single machine scheduling problem where we are given $N$ jobs characterized by release times and deadlines. We are given a processing time and a holding time. In an integer scheduling, each job is assigned a starting time within a constraint set corresponding to an integer number of processing times and holding times. Our goal is to find feasible schedules to alternatively minimize two objectives: the sum of the starting times of all jobs and the makespan (the time at which all jobs are finished). We present approximation algorithms which can alternatively approximate two objectives with factors of 5 and 3, respectively. Our main algorithm consists of solving two subproblems, one of which is solved optimally using dynamic programming, while the other is solved approximately using linear programming relaxation and rounding.

I. INTRODUCTION

Motivation. The almost uninterrupted growth of air traffic in the US in the last decades has motivated the design of a semi-automated Air Traffic Control (ATC) system to help Controllers managing the increasing complexity of traffic flow in the vicinity of major airports. However, some of the crucial tasks of ATC are still performed manually: conflict avoidance, and some aspects of scheduling. In the current ATC system, scheduling arriving flights incoming into busy airports is facilitated by CTAS [11], a software used to compute optimal descent routes for aircraft arriving to destination airports. The combinatorial aspect of incoming traffic optimization in the direct vicinity of airports (i.e. up to 200 nautical miles before landing) has not been addressed in a global manner to this day: the solutions implemented by Controllers result from combinations of procedures established and validated over time with heuristics [5]. The ATC problem which the controllers currently solve manually is the following: each aircraft has an earliest possible arrival time, and a delayed arrival time (if it achieves minor path deviations on the way). The aircraft is allowed to arrive in the interval between these two times. Ideally, these intervals are such that there exists an assignment which guarantees the delivery of one aircraft at most every $\Delta$ time units at the airport. $\Delta$ is sometimes called the metering time. This problem is referred to as the single interval problem. If such an assignment is not possible, ATC decides to put certain aircraft “on hold” in order to meet the $\Delta$ metering constraint of the airport. The choice of aircraft which are put on hold today is not optimized, but obeys experienced-based rules (called playbooks). This problem can be formulated as a single machine scheduling problem as follows. We are given $N$ jobs denoted by $J_1, J_2, \ldots, J_N$. Each job $J_j$ is characterized by two nonnegative numbers $r_j$ and $d_j$. We are given two nonnegative numbers $\Delta$ (processing time), and $T$ (holding time). In a feasible schedule $(i)$ each job $J_j$ is assigned a starting time $\tau_j$, such that $r_j + l \cdot T \leq \tau_j \leq d_j + l \cdot T - \Delta$, for some integer $l \geq 0$ which represents the number of holding patterns used; $(ii)$ if $j \neq j'$ then $|\tau_j - \tau_j'| \geq \Delta$. Our goal is to find a feasible schedule such that one of the two following objectives are minimized: the makespan $C_{\text{max}} = \max_{1 \leq j \leq N} \tau_j + \Delta$ (the time at which all jobs are finished) and the sum of the starting times of all jobs $\sum_{j=1}^{N} \tau_j$.

Related Results. Combinatorial optimization is emerging as a powerful tool for real time systems applications, in particular for Air Traffic Control (see for example [14], [13]). In [5], the authors posed the general aircraft scheduling problem with holding time. They used an integer programming approach to solve the problem, using CPLEX, and showed the limit of this approach (the computational time grows exponentially with the number of aircraft, making this approach unattractive for online implementations). In [6], the authors present a polynomial time algorithm for solving the single interval case: the objective is to maximize $\delta := \min_{1 \leq j < l} |\tau_j - \tau_j'|$ without putting the aircraft on hold. In the general case, $\delta < \Delta$, which
forces ATC to put aircraft on hold: the present algorithm has been developed to address this situation. In [4], we present an implementation of this algorithm for the particular Dallas DFW airport, in a NASA-developed software called TCSim.

Our problem generalizes those studied by Dantzig and Fulkerson [9], and Gertsbakh and Stern [12] where each job’s starting time can only be in a single interval \([r_j, d_j - \Delta]\). To find a feasible schedule for the given jobs, one can schedule them on parallel identical machines. Here, the goal is to minimize the number of machines needed. One can easily see that this problem is equivalent to our problem when the objective is to minimize the number of periods we have to use and when all \(d_j \in [0, T]\). The problem can be solved to optimality if \(r_j = d_j - \Delta\) for all \(j\). If \(r_j < d_j - \Delta\), the complexity of the problem is still open. This work can also be related to the job interval selection problem (JISP) [8], [10].

Motivated by the results of Baptiste [3] that for the single interval case, there exists a polynomial time algorithm that can schedule the maximum number of jobs, we treat the jobs with \(r_j < T\) separately. We thus develop a greedy approach: we schedule the maximum number of jobs in the first period \([0, T]\), and then apply the above mentioned LP-based algorithm to the rest of the jobs. Two issues need to be addressed. (i) After we schedule the jobs in the first interval, how do we bound the optimal cost of the rest of the jobs? Indeed, when we apply the LP-based algorithm to the rest of the jobs, we are comparing their cost in this schedule with their cost if they were scheduled by the optimum solution. Our proof provides a bound on the ratio of these two costs. (ii) Scheduling the maximum number of jobs in the first period does not necessarily imply that the sum of starting times of this set of jobs is minimized. Here, we generalize the algorithm and result of Baptiste [3] and apply it to the set of aircraft which can arrive before \(T\): we derive a polynomial-time algorithm for the single interval problem, which schedules (feasibly) the maximum number of jobs; furthermore, we show that among all feasible schedules that schedule the same number of jobs, the algorithm produces a solution that has the minimum sum of starting times. The algorithm is based on dynamic programming.

Our Results and Techniques. Our first attempt at this problem is based on a simple observation that the starting time of the jobs can be restricted to a polynomial bounded set. Therefore, the problem can be formulated as a constrained assignment problem: assign the jobs to those possible starting points, with the constraints that at most one job can be assigned to any interval of length \(\Delta\). As a standard approach, we solve the linear programming (LP) relaxation of this constrained assignment problem. The key in our algorithm is the rounding of the LP solution \(x\), which is fractional in general, into an integer solution for the original problem. Here we observe that there is a way to modify the fractional solution \(x\) and obtain another fractional solution \(\bar{x}\) (which still satisfies the constraints of the LP) such that the corresponding costs (both makespan and the sum of the starting times) are increased by a factor of at most 3 if all \(r_j \geq T\). Then we construct an instance of the (unconstrained) assignment problem such that \(\bar{x}\) is a feasible solution of the assignment problem and any integer solution of the new instance will automatically satisfy the constraints of the original problem. Using the fact that we can find an integer solution for the (unconstrained) assignment problem in polynomial time, whose cost is not worse than that of \(\bar{x}\), we find an approximation solution for the original problem with guaranteed error factor if all \(r_j \geq T\). However, bounding the ratio of the algorithm if there are jobs with \(r_j < T\) requires an initial step to schedule these jobs.

1 However, because of the structure of our problem, it seems that none of the algorithms can be applied to give an approximation for our problem. In most of the models, only release times of jobs are present, but here, each job has a deadline, although the job can be scheduled in a prescribed later range if this deadline cannot be met.
Combining the above algorithms, we obtain an approximation algorithm which approximates the optimal solution of minimizing the sum of starting times with factor 5. For makespan, we modify both parts of the algorithm and obtain a ratio 3.

This article is organized as follows. Section II describes the optimization program formulation of the physical problem. Section II-A shows a trivial 2 approximation algorithm for the case in which \( T \leq \Delta \), which works when the objective function is sum of arrival times or makespan. Section II-B provides a high level description of the approximation algorithm for the sum of arrival times in the case \( T > \Delta \). The part of the algorithm which schedules the first set of jobs before \( T \) (dynamic programming) is explained in detail in Section III. Section IV presents the part of the algorithm which schedules the rest of the jobs after \( T \) (LP rounding) and explains how to modify the algorithm to provide a 3-approximation algorithm for makespan.

II. HIGH-LEVEL DESCRIPTION

We call \( a_i \) the earliest arrival time of aircraft \( i \), and \( b_i \) the latest arrival time of aircraft \( i \) without holding pattern. In the absence of holding pattern, the problem of scheduling aircraft \( i \) in \([a_i,b_i]\) for all \( i \leq N \) such that two aircraft are separated by at least \( \Delta \) is equivalent to scheduling \( N \) jobs on a single machine, with processing times \( \Delta \), release times \( r_i = a_i \), deadlines \( d_i = b_i + \Delta \). We call \( T \) the time of a holding pattern, and let \([a_j,b_j]+TN := \bigcup_{k=0}^{\infty} \{a_j + kT, b_j + kT\} \). The problem can be formulated as follows:

\[
\begin{align*}
\text{min:} & \quad \sum_{j \in \{1, \ldots, N\}} \tau_j \quad \forall j \in \{1, \ldots, N\} \\
\text{s.t.:} & \quad \tau_j \in [a_j, b_j] + TN, \\
& \quad |\tau_j - \tau_{j'}| \geq \Delta \quad \forall (j, j') \in \{1, \ldots, N\}^2, \\
& \quad \text{such that } j \neq j' 
\end{align*}
\]

A. A Trivial Algorithm for the \( T \leq \Delta \) Case

**Theorem 1.** The first come first served algorithm below is a 2 approximation algorithm for makespan and sum of arrival times.

1. Sort the aircraft by earliest possible time of arrival: without loss of generality, we can write \( a_1 \leq a_2 \leq \cdots \leq a_{N-1} \leq a_N \).

2. Construct the following schedule:

\[
\begin{align*}
\tau_1 & = a_1 \\
\tau_j & = \min \{ x \mid x \in [\tau_{j-1} + \Delta, +\infty) \cap ([a_j, b_j] + TN) \} \quad \forall j \in \{2, \ldots, N\}
\end{align*}
\]

**Proof of Theorem 1:** Sum of arrival times: Divide the \( N \) aircraft into \( K \) subsets: \( S_1 = \{N_1, \ldots, N_2 - 1\} \), \( S_2 = \{N_2, \ldots, N_3 - 1\} \), \( S_K = \{N_K, \ldots, N\} \), where \( N_1 = 1 \)

\[
N_k = \min \{ p | a_{N_k-1} + (p - N_k - 1)(\Delta + T) < a_p \}
\]

For all \( k \in \{1, \ldots, K-1\} \), for all \( j \in S_k \), the algorithm provides \( \tau_j \leq a_{N_k} + (j - N_k)(\Delta + T) \). Indeed, if it were not true, we would have a \( j \in S_k \) such that \( \tau_j > a_{N_k} + (j - N_k)(\Delta + T) \), which would contradict the definition of \( N_{k+1} \). We call \( c \) the cost of this algorithm:

\[
c = \sum_{k=1}^{K} \sum_{j \in S_k} \tau_j
\]

\[
\leq \sum_{k=1}^{K} \sum_{j \in S_k} \left( a_{N_k} + (j - N_k)(\Delta + T) \right)
\]

\[
= \sum_{k=1}^{K} \sum_{j \in S_k} \frac{(a_{N_k} + \sum_{i=1}^{j} (j - N_k)\Delta)}{a_{N_k} + (j - N_k)\Delta}
\]

Since we know that \( \text{OPT} \geq \sum_{k=1}^{K} \sum_{j \in S_k} (a_{N_k} + (j - N_k)\Delta) \), we get the following approximation ratio \( \beta \) for the sum of arrival times:

\[
\beta = \max_{k=1}^{K} \max_{j \in S_k} \left( \alpha \right)
\]

\[
= \max_{k=1}^{K} \max_{j \in S_k} \left( \frac{a_{N_k} + \sum_{i=1}^{j} (j - N_k)\Delta}{a_{N_k} + (j - N_k)\Delta} \right)
\]

\[
\leq 1 + \frac{\Delta}{2} \leq 2
\]

**B. The Case \( T > \Delta \)**

Before presenting our algorithm, we introduce some notation and initial steps.

- Sort the aircraft by earliest possible time of arrival: without loss of generality, we can write \( a_1 \leq a_2 \leq \cdots \leq a_{N-1} \leq a_N \).
- Divide the \( N \) aircraft into \( K + 1 \) subsets: \( S_0 = \{N_0, \ldots, N_1 - 1\} \), \( S_1 = \{N_1, \ldots, N_2 - 1\} \), \( \cdots \), \( S_K = \{N_K, \ldots, N\} \) where \( N_0 = 1 \), and \( N_k \) is given by \( N_k = \min \{ p | a_{N_{k-1}} + (p - N_k - 1)\Delta < a_p \} \).
- For each \( k \), index the elements \( t_i \) of \( \sum_{k=1}^{K} \sigma_k \) by \( i \in \{1, \ldots, |\Sigma|\} \) in increasing order.

Step (i) is used for notational convenience. Step (ii) enables us to construct a grid of polynomial size: it separates aircraft into groups for which we have at least a feasible solution (which we can construct using the trivial algorithm of Section II-A). In step (iii), the \( K \) groups are gridded \( \sigma_k \) and assembled in a single grid \( \Sigma \) of size \( O(\frac{\Delta}{T}N^2) \). The set \( \bigcup_{k \in S_k} \{a_i + \Delta N + TN\} \) is the set of earliest arrival times plus an integer number of \( T \) (holding patterns) and/or \( \Delta \) (aircraft separation). It follows from Proposition 1 that
Main Algorithm

1) If \( a_1 < T \), call \( F \) the set of \( i \) such that \([a_i, b_i] \cap \{t \leq T \} \) is not empty. Call \([a_i, b_i] = [a_i, b_i] \cap \{t \leq T \} \). Schedule the maximum number of aircraft of \( F \) according to the algorithm of Section III. If this number is equal to \( N \), stop.

2) Solve the relaxed LP (3) for the remaining aircraft. If \( a_1 \geq T \), solve the relaxed LP (3) directly.

Minimize: \[
\sum_{j} \sum_{i \in G(j)} t_i x_{ij} \\
\sum_{i \in G(j)} x_{ij} = 1 \quad \forall j \in \{1, \ldots, N\} \\
x_{ij} \geq 0 \quad \forall j \in \{1, \ldots, N\}, \forall i \in G(j) \\
\sum_{i \in I(i)} x_{ij} \leq 1 \\
\forall \in \{1, \ldots, |I|\}
\]

where \( \forall i \in \{1, \ldots, |I|\}, I(i) = \{ j' \leq |I| \mid t_{ij'} \geq t_i \wedge t_{ij'} - t_i < \Delta \}, \) and \( \forall j \in \{1, \ldots, N\}, G(j) = \Sigma \cap \{[a_j, b_j] + TN\} \).

3) Modify the \( x_{ij} \) using the procedure \( T(x_{ij}) = \text{TransformLPSol}(x_{ij}) \).

4) Compute the integer solution \( x_{ij} = \text{Matching}(\pi_{ij}) \) of the matching problem constructed by the Matching procedure. The result is a feasible schedule for the remaining aircraft.

the optimum schedule lies on this grid (the optimum is sometimes referred to as left shifted).

In the Main Algorithm shown next page, \( I(i) \) represents the set of aircraft for which the arrival times are less than \( \Delta \) time units after \( t_i \), and thus not allowed if \( t_i \) is chosen by an integer solution of (3), \( G(j) \) represents the set of \( i \) such that \( t_i \) is available for aircraft \( j \). We call \( n \) the number of aircraft scheduled before \( T \) by the algorithm of Step 1 of the Main Algorithm, and \( m = N - n \) the number of aircraft scheduled after \( T \). We use the following notation:

- \( C(n) \): cost of scheduling \( n \) jobs before \( T \) with Step 1 of the Main Algorithm (cost means sum of arrival times)
- \( C(LP, m) \): min. cost of relaxed LP (3) for the \( m \) jobs
- \( C(HP, m) \): min. cost of a feasible integer solution to (3) for the \( m \) jobs
- \( C(\pi, m) \): cost of the fractional solution for the \( m \) jobs \( \pi_{ij} \) of (3) produced by Step 3 of the Main Algorithm
- \( C(\bar{x}, m) \): cost of the fractional solution for the \( m \) jobs \( \bar{x}_{ij} \) produced by Step 4 of the Main Algorithm
- \( OPT(m) \): cost of the \( m \) jobs when they are scheduled by the optimal solution OPT
- \( OPT(n, \text{first}) \): cost of the first \( n \) jobs when they are scheduled by OPT
- \( OPT(m, \text{last}) \): cost of the last \( m \) jobs when they are scheduled by OPT

**Lemma 1.** Consider the \( |F| \) aircraft of \( F \) (Step 1 of the Main Algorithm). Step 1 schedules the largest number of them in \( \bigcup_{i=1}^{N} [a_i, b_i], \) with minimum sum of arrival times. The complexity of Step 1 is \( O(|F|^9) \).

**Lemma 2.** The sum of starting times for the \( m = N - n \) other aircraft scheduled after \( T \) is bounded above by \( 5 \cdot OPT(m, \text{last}) \). The complexity of scheduling these \( m \) aircraft is dominated by solving the LP (3).

**Theorem 2.** The Main Algorithm has a 5-approximation ratio for the sum of arrival times. Its complexity is \( O(N^9) \), and is dominated by Step 1 of the Main Algorithm.

**Proof of Theorem 2:** Let \( c_n \) be the cost of the \( n \) jobs scheduled before \( T \) by Step 1. Let \( c_m \) be the cost of the \( m \) jobs scheduled after \( T \) by Step 4. By Lemma 1, we have \( c_n \leq OPT(n, \text{first}) \). By Lemma 2, we have \( c_m \leq 5 \cdot OPT(m, \text{last}) \). Now combining the two bounds for \( c_n \) and \( c_m \), we get:

\[
c_n + c_m \leq OPT(n, \text{first}) + 5 \cdot OPT(m, \text{last}) \leq 5 \cdot OPT
\]

The complexity of Step 1 is \( O(|F|^9) \). The worst case is when \( |F| = N \) (all jobs can be scheduled in \( F \)), therefore the complexity of Step 1 is \( O(N^9) \). The complexity of the other steps is determined by solving the linear program (3), so the overall complexity is \( O(N^9) \).

**III. Dynamic Programming Algorithm for Step 1 of the Main Algorithm**

We call \( |F| = f \in \mathbb{N} \). We are given a set of time intervals \( [r_i, d_i], i \in \{1, \ldots, f\} \), and assume that the \( d_i \) have been sorted in chronological order: \( d_1 \leq d_2 \leq \ldots \leq d_f \). A schedule \( \{t_i\}_{i=1}^{f} \) is said to be admissible if for all \( i \) in \( \{1, \ldots, f\} \), \( t_i \geq r_i \) (jobs start after their release time), \( t_i + \Delta \leq d_i \) (jobs are finished before their deadline), and \( \forall i \neq j, |t_i - t_j| \geq \Delta \) (jobs are separated by \( \Delta \)). For \( l \) given \( l \leq f \), we want to compute \( \min \sum_{p=1}^{l} t_{ip} \), the minimum of the sum of starting times of \( l \) of the \( f \) jobs.

**Definition 1.**
- We call \( \Theta = \{t \mid \exists i \in \{1, \ldots, f\}, \exists l \in \{0, \ldots, f\}, \text{ such that } t = r_i + l \Delta \} \).
- We call \( U_k(s, e) = \{J_i \mid i \leq k \text{ and } s \leq r_i < e\} \) the set of jobs of index less than \( k \) released between \( s \) and \( e \).
- For a given \( l \), we call \( C_k(s, e, l) \) the minimum sum of starting times among the set of admissible schedules containing \( l \) jobs in \( U_k(s, e) \), which satisfy
  - (i) for all \( i \), \( t_i \geq s + \Delta \) (job \( i \) starts after \( s + \Delta \))
  - (ii) for all \( i \), \( t_i + \Delta \leq e \) (job \( i \) is processed before \( e \))
  - (iii) for all \( i \), \( t_i \in \Theta \) (starting times are in \( \Theta \)).

We pose \( C_k(s, e, l) = +\infty \) for all \( l > k \).

**Proposition 1.** There exists an optimal schedule such that \( t_i \in \Theta \) for any \( i \) scheduled.
The proof is an extension of Carlier [7], Baptiste [3].

**Proposition 2.** $C_k(s, e, l)$ can be computed recursively by the following formula:

$$C_k(s, e, l) = C_{k-1}(s, e, l)$$

if $r_k \notin [s, e]$ for the $k^{th}$ job $J_k$, and

$$C_k(s, e, l) = \min \left\{ \begin{array}{c} C_{k-1}(s, e, l) \\ \min (C_{k-1}(s, s', l - q) + s' + C_{k-1}(s', e, q - 1)) \\ 1 \leq q \leq l - 1 \\ \max (r_k, s + \Delta) \leq s' \leq \min (d_k, e) - \Delta \end{array} \right\}$$

otherwise.

**Proof of Proposition 2:** Please see Appendix.

**Proof of Lemma 1:** The aircraft scheduling problem can be transformed into a job scheduling problem by letting $r_i = a_i$ and $d_i = b_i + \Delta$, for $i \in I$, as defined in the Main Algorithm. We call $f = |I|$. For notational convenience, we can relabel these jobs from 1 to $f$. Note that $f$ can range from 1 to $N$. We can now apply Proposition 2. For all $k$, $s$, $e$, $l$, such that

$$k \leq f$$

$$s \geq \min_{i=1}^{f} r_i - \Delta$$

$$e \leq \max_{i=1}^{f} d_i$$

$$l \leq f$$

one can compute recursively $C_k(s, e, l)$, and find $l_0$, the largest $l$ such that $C_f(\min_{i=1}^{f} r_i - \Delta, \max_{i=1}^{f} d_i, l_0) = 1$.

$k$ and $l$ range from 1 to $f$, $e$ and $s$ are on the grid $\Theta$ of Proposition 1, thus they are in a set of size $O(f^2)$. The size of the dynamic program table to build is thus $O(f^6)$. In building the table, we take the minimum over a set of size $O(f^3)$, since it is indexed by $s' \in \Theta$ and $q \in \{1, \ldots, l - 1\}$. Thus, the total cost of the algorithm is $O(f^9)$.

**IV. LINEAR PROGRAMMING BASED ALGORITHM FOR STEPS 2-3-4 OF THE MAIN ALGORITHM**

This section summarizes steps 2-3-4 of the Main Algorithm. We first solve the relaxed LP (3). This LP is a constrained assignment problem: aircraft $j$ is assigned arrival times $t_j$; the last constraint means that any integer solution of this problem can only assign one $t_j$ every $\Delta$ (by definition of the interval $I(i)$). In the relaxed version, it means that the $x_{ij}$ (fractional assignments) sum to at most 1 over a period of $\Delta$.

The fractional solution $\bar{x}$ is transformed to another fractional solution $\bar{\pi}$ by the procedure TransformLPsol (illustrated in Figure 1). The fractional solution is decomposed into sets of $t_i$ such that $t_i \in [qT, (q + 1)T)$, where $q \in \mathbb{N}$. Each of these sets is decomposed into chunks of length $\Delta$. Figure 1 shows one of these intervals (eight chunks of length $\Delta$, and the last chunk of length less than $\Delta$). The $x_{ij}$ in each of these chunks is shifted by $(2q - 2)T$ or $(2q - 1)T$ depending on parity, in order to insert idle time of length $\Delta$ between the corresponding chunks (see arrows in Figure 1). Because of the last chunk (interval $[t_{i,q}, \ldots, t_{i,q+1}]$ in Figure 1), it is necessary to insert a period of idle time of length at least $\Delta$ after the highest shifted $x_{ij}$ (the black chunk coming from $[t_{i,q}, \ldots, t_{i,q+1}]$ in Figure 1). This is done by adding a full interval $[3qT, (3q + 1)T]$ of idle time after $3qT$. This is not optimal, but enables us to construct $\bar{\pi}$ systematically.

The result is another fractional solution $\bar{\pi}$ which satisfies (3) and has idle time periods of length $\Delta$ alternating with non idle time periods of length $\Delta$. This means sets in which $x_{ij} = 0$ for $t_i$ ranging in an interval of length $\Delta$ alternate with sets of the same length with nonzero $x_{ij}$.

The procedure Matching takes the new fractional solution $\bar{\pi}$, and constructs a feasible instance of a weighted assignment problem. An illustration is given in Figure 2. Every chunk $\{i_{m}, \ldots, i_{m+1}, \Delta - 1\}$ with nonzero $x_{ij}$ is reduced to a single node indexed by $r$. Any aircraft $j$ which has one (or more) nonzero $x_{ij}$ with $i \in \{i_{m}, \ldots, i_{m+1}, \Delta - 1\}$ is now linked to $r$ (Figure 2 right). The weight on the corresponding link is the smallest $t_i$ with nonzero $x_{ij}$ in $\{i_{m}, \ldots, i_{m+1}, \Delta - 1\}$. For example in Figure 2 right, the weight on the link $r \rightarrow (j - 1)$ is $t_{27}$ because in Figure 2 left, node $j - 1$ was such that $x_{27}, x_{29}$ and $x_{35}$ are nonzero. The corresponding weighted assignment problem is (4). A fractional feasible solution $\bar{x}$ is obtained from $\bar{\pi}$ by adding all $x_{ij}$ emanating from the same chunk towards aircraft $j$. We know that (4) has an integer optimal, which is therefore less or equal to our fractional feasible solution. We can now prove Lemma 2 (notations are defined on page 5).

**Proof of Lemma 2:** Let us call $e_m$ the cost of the $m$ jobs scheduled after $T$. For each of these jobs, we compute an upper bound of the ratio by which the cost is increased by the procedure TransformLPsol for each nonzero $x_{ij}$. The minimum cost $C(LP, m)$ of scheduling the $m$ jobs after $T$ is less than the cost of scheduling these $m$ jobs with OPT and shifting them by $T$. Therefore:

$$C(LP, m) \leq C(IP, m) \leq OPT(m) + mT$$

Let us define $B_q$ as the sum of fractional $x_{ij}$ in interval $[qT, (q + 1)T)$:

$$B_q = \sum_{i\in t_{i,q}} \sum_{i\leq j < (q+1)T} x_{ij}.$$

Let us call $r$ the largest $q$ such that $B_q \neq 0$. By construction of the LP (3), we have

$$\sum_{q=1}^{r} B_q = m \text{ and } \sum_{q=1}^{r} B_q \cdot qT \leq C(LP, m)$$

Let us define $C(B_q)$ as the cost of the fractional LP solution of (3) corresponding to the interval $[qT, (q + 1)T)$. As shown in Figure 1, the amount by which $x_{ij}$ is shifted is either $(2q - 2)T$ or $(2q - 1)T$, and therefore the increase in cost of the $x_{ij}$ in interval $[qT, (q + 1)T)$ due to the procedure TransformLPsol is
Procedure TransformLPsol

Input: fractional solution $x_{ij}$ of (3).
Output: $\pi_{ij}$, another feasible fractional solution of (3).

Perform the following assignment:

- $\pi_{ij} = x_{ij} \text{ where } t^i = t^i_{ij} + (2q - 2)T$, if $3q \geq 0, \exists k > 0$, such that $l < \frac{t^i_{ij}}{T}$ is even, and $t^i_{ij} \in [qT + l\Delta, qT + (l + 1)\Delta)$ or such that $l = \frac{t^i_{ij}}{T}$ is even, and $t^i_{ij} = t^i_{ij} + (2q - 2)T$ for $t^i_{ij} \in [qT + l\Delta, (q + 1)\Delta)$.
- $\pi_{ij} = x_{ij} \text{ where } t^i = t^i_{ij} + (2q - 1)T$, if $3q \geq 0, \exists k > 0$, such that $l < \frac{t^i_{ij}}{T}$ is odd, and $t^i_{ij} \in [qT + l\Delta, qT + (l + 1)\Delta)$ or such that $l = \frac{t^i_{ij}}{T}$ is odd, and $t^i_{ij} = t^i_{ij} + (2q - 1)T$ for $t^i_{ij} \in [qT + l\Delta, (q + 1)\Delta)$.

The two last items correspond to the last (incomplete) chunk of $[qT, (q + 1)T].$

Fig. 1. Illustration of the procedure TransformLPsol. This procedure transforms the feasible fractional solution $x = x_{ij}$ of (3) into another feasible fractional solution $\pi = \pi_{ij}$ with idle time. Half of the fractional $x_{ij}$ in the interval $[qT, (q + 1)T]$ is shifted to $[(3q - 2)T, (3q - 1)T]$ (shift by $(2q - 2)T$), and the other half to $[(3q - 1)T, 3qT]$ (shift by $(2q - 1)T$). The interval $[3qT, (3q + 1)T]$ is empty for now; this is a waste of space, but this enables us to satisfy the constraints of (3) for the part of the fractional solution between $\frac{t^i_{ij}}{T}$ (i.e. on the figure) and $\frac{t^i_{ij}}{T} + 1$.

Fig. 2. Illustration of the procedure Matching. This procedure transforms the feasible fractional solution $\pi_{ij}$ into a weighted matching problem. In the fractional LP solution, only the nonzero $\pi_{ij}$ are represented (solid if they are in the range of the figure, dashed if they connect aircraft with arrival times outside the figure). On the right plot, the weights are the arrival times from the fractional LP solution: for example, $\Theta_{rj - 1} = t_{27}$. 
C(\pi, m) \leq \sum_{q=1}^{T} (C(B_q) + B_q \cdot (2q - 1)T) \\
= \frac{C(LP, m)}{C(T, m)} \cdot \sum_{q=1}^{T} B_q \cdot (2q - 1)T \\
\leq \text{OPT}(m) + mT + \sum_{q=1}^{T} B_q \cdot (2q - 1)T \\
= \text{OPT}(m) + 2 \sum_{q=1}^{T} B_q \cdot (2q - 1)T \\
\leq \text{OPT}(m) + 2 \sum_{q=1}^{T} B_q \cdot (2q - 1)T \\
\leq \text{OPT}(m) + 2B_T \cdot mT \\
\leq \text{OPT}(m) + 5 \cdot B_T \cdot mT \\
\leq \text{OPT}(m, \text{last})
Appendix: Proof of Proposition 2

Case 1: \( r_k \notin [s,e] \)
We have \( J_k \notin U_k(s,e) \) by definition of \( U_k(s,e) \). We know that \( U_k(s,e) \subseteq \cap_{k=1}^{l} U_{k-1}(s,e) \). Since \( J_k \notin U_k(s,e) \), this implies \( U_k(s,e) \subseteq \cap_{k=1}^{l} U_{k-1}(s,e) \). The reverse inclusion \( \cap_{k=1}^{l} U_{k-1}(s,e) \subseteq U_k(s,e) \) holds by definition of \( U_k(s,e) \), and we therefore have \( U_k(s,e) = \cap_{k=1}^{l} U_{k-1}(s,e) \). By definition of \( C_k(s,e,l) \), this implies that \( C_k(s,e,l) = C_{k-1}(s,e,l) \).

Case 2: \( r_k \in [s,e] \)
We have \( J_k \in U_k(s,e) \). We call
\[
C' = \min \left\{ C_{k-1}(s,e,l), \right\}
\]
we prove that \( C' = C_k(s,e,l) \) in two steps.

Step 1: \( C' \geq C_k(s,e,l) \)

- If \( C' = C_{k-1}(s,e,l) \), the inclusion \( U_{k-1}(s,e) \subseteq U_k(s,e) \) implies \( C' = C_{k-1}(s,e,l) \geq C_k(s,e,l) \).
- If \( C' = C_{k-1}(s',s',l-q)+s'+C_{k-1}(s',e,q-1) \) for some \( s' \) and \( q \) achieving the min, we separate the jobs into three subsets: \( X \), the \( l-q \) first jobs; \( J_k \); and \( Y \), the set of \( q-1 \) jobs after \( s' \).

We therefore have \( X \cup Y \cup \{J_k\} \subseteq U_k(s,e) \).

1. \( \forall j \in J \subseteq U_{k-1}(s,s') \subseteq U_k(s,e) \subseteq U_k(s,e) \).
2. \( \forall j \in Y \subseteq U_{k-1}(s',s') \subseteq U_k(s',e) \subseteq U_k(s,e) \).
3. \( J_k \subseteq U_k(s,e) \) by assumption.

Therefore, \( X \cup Y \cup \{ J_k \} \subseteq U_k(s,e) \), and \( |X| + |Y| + 1 = l-1+q-1-1=l \). From this, we see that \( X \cup Y \cup \{ J_k \} \) is a particular element of the set among which the min is taken in the definition of \( C' \) above. Therefore, \( C' \geq C_k(s,e,l) \) which is the min among all elements of this set.

Step 2: \( C' \leq C_k(s,e,l) \)

Call \( B \) the instantiation of the \( l \) jobs which realizes \( C_k(s,e,l) \).

- If \( J_k \notin B \), then \( B \subseteq U_k(s,e) \cup \{ J_k \} \subseteq U_{k-1}(s,e) \), and \( |B| = l \). \( B \) is also an instantiation of \( C_{k-1}(s,e,l) \), therefore \( C_k(s,e,l) \geq C_{k-1}(s,e,l) \). By definition of \( C' \), we have \( C_{k-1}(s,e,l) \geq C' \).

Thus, \( C_k(s,e,l) \geq C' \).

- If \( J_k \in B \), we first show that \( \exists j \in Y, r_j > s' \) for \( s' \) the starting time of job \( J_k \).

Let \( j \) be in \( Y \). Assume \( r_j \leq s' \). We know that \( r_j \leq s' \) by construction. We know that \( j \leq k \), because \( J_k \in Y \) and \( Y \subseteq B \subseteq U_k(s,e) \); in \( U_k(s,e) \), all jobs have index less than \( k \). By assumption, the deadlines have been indexed chronologically, therefore \( d_j \leq d_k \). Call \( z \) the completion time of job \( j \). We have \( d_j \geq z \). Summarizing all information above, this means that both \( J_k \) and \( J_k \) can start at \( s' \) and finish at \( z \). Therefore, switching \( J_k \) and \( J_k \) is possible and will not change the sum of the starting times of all jobs. The conclusion of this paragraph is that any job of \( Y \) with release time less than \( s' \) can be put in \( X \) without change of cost. We can therefore assume that all jobs \( J \) of \( Y \) have a starting time \( r_j > s' \).

Let \( J_k \) be in \( Y \). Because \( J_k \in U_k(s,e) \), \( j \leq k \). Since \( J_k \in Y \), \( J_k \) is scheduled after \( J_k \) by the previous paragraph. Therefore, \( J_k \leq J_k \). This implies \( J_k \in U_{k-1}(s',e) \). Therefore \( Y \subseteq U_{k-1}(s',e) \). Let us call \( C(Y) \) the cost of scheduling all jobs of \( Y \) \( C(Y) \) is the sum of their starting times. \( C(Y) \geq C_{k-1}(s',e,|Y|) \) by definition of \( C_{k-1}(\cdot,\cdot,|\cdot|) \). For \( X \), similarly, \( J_k \in X \subseteq B \subseteq U_k(s,e) \) implies \( r_j \geq s \).

Also, \( r_j < s' \) because \( r_j \leq s' - \Delta \), \( j \neq k \) because \( J_k \in X \), therefore \( j \leq k - 1 \). From this we deduce \( X \subseteq U_{k-1}(s',s') \). This implies \( C(X) \geq C_{k-1}(s',s',|X|) \) where \( C(X) \) is the cost of scheduling the jobs of \( X \).

\[
C(X) \geq C_{k-1}(s',s',|X|)
\]
\[
C(Y) \geq C_{k-1}(s',e,|Y|)
\]

Therefore, writing explicitly the contributions of the different terms in \( B \), we have:

\[
C_k(s,e,l) = C(X) + s' + C(Y) \geq C_{k-1}(s',s',|X|) + s' + C_{k-1}(s',e,|Y|) \geq C'
\]

The last inequality results from the min in the definition of \( C' \). □