Runway Sequencing with Holding Patterns

Konstantin Artiouchine [*] Philippe Baptiste [†] Christoph Dürr [‡]

October 28, 2004

Abstract

We study a scheduling problem, motivated by air-traffic control, in which a set of aircrafts are about to land on a single runway. When coming close to the landing area of the airport, a set of time windows in which the landing is possible, is automatically assigned to each aircraft. The objective is to maximize the minimum time elapsed between any two consecutive landings. We study the complexity of the problem and describe several special cases that can be solved in polynomial time. We also provide a compact Mixed Integer Programming formulation that allows us to solve large instances of the general poblem when all time windows have the same size. Finally, we introduce a general hybrid branch and cut framework, based on Constraint Programming and Mixed Integer Programming, to solve the problem with arbitrary time windows. Experimental results are reported.
Keywords: Air-traffic control, Scheduling, Hybrid search

1 Aitraffic Control in the TRACON area

Airport arrival and departure management is a very complex problem that plays a crucial role for airports. As both the number of runways and the number of air trafic controllers are limted, the trafic has to be carefully planed to limit peaks of activity (take off and landing) while matching as much as possible airlines landing times requirements. This planing problem, known as the "Time slot Allocation" problem has been widely studied in the literature (see [3] for a review). Good plannings allow airports to reduce delays while keeping safety standards high.

[*]Thales TRT, Domaine de Corbeville, 91404 Orsay CEDEX, France and LIX, Ecole Polytechnique, 91128 Palaiseau, France

[†]LIX, Ecole Polytechnique, 91128 Palaiseau, France

[‡]LRI, Université Paris-Sud, 91405 Orsay, France

However unpredictable delays make it hardly impossible to precisely schedule aircrafts in advance. So the initial planning has to be refined on line when aircrafts are close enough to the airport, *i.e.*, when they reach the final descent in the TRACON (Terminal Radar Approach CONontrol). Typically, the TRACON controls aircraft approaching and departing between 5 and 50 miles of the airport. In this final scheduling process, air traffic controllers make some aircrafts wait before landing. Unfortunately this "waiting" process is complex as aicrafts follow predetermined routes and their speed cannnot change much (speed is heaviely constrained by the type of aicraft, the altitude, the weather, *etc.*). To reach some degree of flexibility in the process, two kinds of delaying procedures are used (see [4] for details):

- First, the speed of the aircratf can be slightly decreased to increase the arrival time. Second, the flight plan can be lengthened by a "Vector For Spacing": Instead of following the the shortest route between two points, the aircraft makes a turn with a small rotation.

  In both cases, this allows air traffic controllers to slightly increase the time taken by an aircraft to fly from one point to another.

- "Holding Patterns" generate a constant prescribed delay for an aicraft. Several holding patterns may exist in the same TRACON (see Figure 1 for an example) and an aicraft can enter such an holding pattern several times.

  When more than one aircraft is holding at the same place, it is often called a "stack". The first aircraft entering the stack is cleared to the lowest practical altitude and subsequent aircraft are cleared to the next available flight level. When aircraft are "peeled off the stack", they are cleared out of holding from the bottom, and new aircraft enters at the top of the stack. As aircraft are descended within the stack, as the bottom Flightlevel is vacated, higher aircraft can be cleared to the next lower available altitude, when a lower aircraft reports leaving that Flightlevel.

Holding patterns are usually not an option of choice for air traffic controllers and their objective is to reduce the number of holding pattrens.

Relying on delaying mechanisms described above, a set of time windows in which the landing is possible can be associated to each aircraft entering the TRACON area. Each time window corresponds to a combination of holding patterns together with some vectors for spacing. As holding patterns usually induce a larger delay than Vectors for Spacing, we can assume that each interval corresponds to a holding pattern (except the first one). Computing all these windows is out of the scope of this paper but this proble is adressed for instance in XXX.

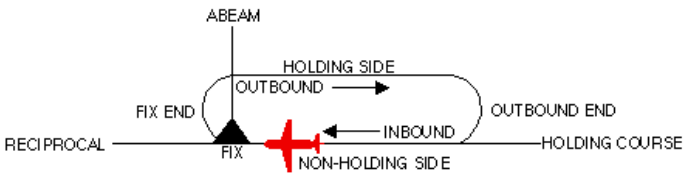

Figure 1: A simple Holding Pattern as described in pilots manuals.

In this paper we study the airport arrival problem with a single runway and we consider two objective functions:

- Minimizing the maximal number of times a plane enters a Holding Pattern.
- Maximizing the minimum time elapsed between two consecutive landings.

We first describe a precise model of this problem (Section 2) initially introduced by Bayen and others [5] and we provide some complexity results. We then describe a branch and cut procedure based on constraint programming and mixed integer programming. We show that this formulation outperforms the pure Mixed Integer Programming formulation proposed in [4].

2 Problem Definition

We assume there are n aircrafts in the TRACON. When entering this area, a set of n_u disjoint time intervals $\cup_{u=1}^{n_i} I_{iu}$ is assigned to each aircraft i; it corresponds to the possible landing times (taking into account possible holding patterns and all possible speed modifications). We consider two objective functions:

- Maximizing the minimum time elapsed between any two consecutive landings.
- Assuming a minimal time p during any two consecutive landings, minimizing the maximal number of times a plane enters a holding pattern.

Note that the decision variants of these two problems are exactly identical. The corresponding problem can then be seen as a simple single machine scheduling problem in which n "landing" jobs with identical processing time p have to be scheduled within some time windows on a single "runway" machine. More formally, the problem can be described as follows.

THE RUNWAY SCHEDULING PROBLEM

input integers $n, p, (s_1, \ldots, s_n), (r_1^1, d_1^1, \ldots, r_1^{s_1}, d_1^{s_1}), \ldots, (r_n^1, d_n^1, \ldots, r_n^{s_n}, d_n^{s_n})$.

meaning each job i has processing time p and has to be fully scheduled (*i.e.*, started and completed) in one of the intervals $[r_{iu}, d_{iu}]$. We wish to find a schedule such that every job is scheduled non-preemptively, and no two jobs overlap.

output a set of starting times $S_1, \ldots, S_n \in \mathbb{N}$ such that (1) $\forall i \in \{1, \ldots n\}$, $\exists j \in \{1, \ldots, s_i\}$ such that $S_i \in [r_{ij}, d_{ij} - p]$ and (2) $\forall i, k \in \{1, \ldots n\}$ with $k \neq i$, $|S_i - S_k| \geq p$.

Figure 2 illustrates a special case of the runway scheduling problem where time windows and diatnces between time windows are identical. 8 aircrafts with 3 time windows are to land on the runway. The shaded bars represent a feasible solution.

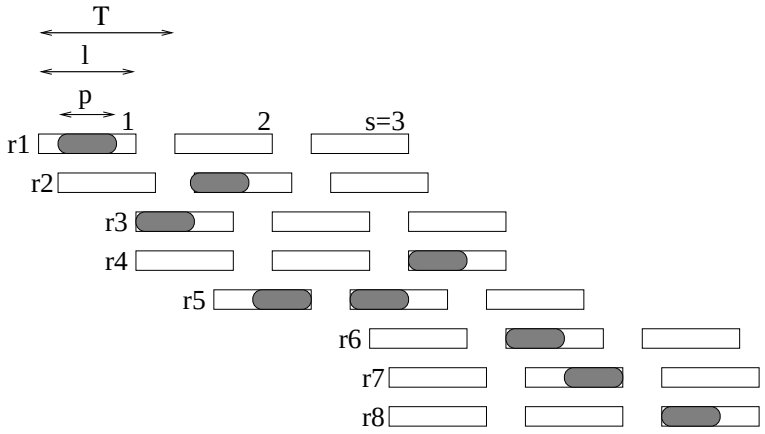

Figure 2: Illustration of the Runway problem.

In the following, we study special cases of the Runway Scheduling Problem in which there is a single pattern for the time windows: same number s of windows per aircraft, same window size l and identical distances T between windows (see Figure 2).

$$\begin{cases} \forall i \leq n, s_i = s \\ \forall i \leq n, \forall j \leq s, d_{ij} - r_{ij} = l \\ \forall i \leq n, \forall j \leq s, r_{ij} = r_i + (j-1)T \end{cases}$$

This problem is reefered to as the "MONO-PATTERN RUNWAY SCHEDULING PROBLEM". It plays a special role since it corresponds to the situattion where all aircraft are identical and where a single holding patter is considered.

To our knowledge the problem has first been studied by Bayen, Tomlin, Ye and Zhang [6], when a single time window only is allowed and the goal is to maximize

the minimal distance between two consecutive landings. As noticed by the authors, the problem is a special case of the decision variant of $1|r_i, p_i = p|L_{\max}$ and hence can be solved in polynomial time (see [9]).

In [5] a slightly more general problem is considered. Holding Patterns are allowed and the time windows do not have the same lengths. The objective is either to minimize the total waiting time for aircraft or to minimize the latest landing time. Constants 5 and 3 approximation algorithms are given for these objectives. Finally, let us mention that a general MIP formulation is provided in [4]. This formulation is carefully reviewed in Section 4.

In Section 3 we prove that the problem is NP-Hard and we review some polynomially solvable cases. Finally, we compare a pure Mixed Integer Programming formulation introduced by Bayen and Tomlin to a Branch and Cut procedure (Section 4). Computational experiments show that our approach compares well to the pure MIP.

3 On the Complexity of the Runway Scheduling Problem

The Runway Scheduling Problem is NP-Complete as soon as processing times are larger than 2 and when we have more than three time windows per aircraft. Indeed, when $p = 2$, $\forall i, s_i = 3$ and $\forall i \leq n, \forall j \leq s, d_{ij} - r_{ij} = 2$, this problem is a direct generalization of the problem of scheduling short tasks with few starting times [14]. Note also that the more general problem, where each job i has its own processing time p_i is also NP-complete, even if all release dates are 0, and $s = 2$.

In the following, we list the special cases that we can solve in polynomial time.

3.1 Unit Processing time

This problem can be reduced to a well-known flow problem in a bipartite graph $G = (\mathcal{J} \cup \mathcal{T}, E)$ where $\mathcal{J}$ is the set of job vertices $J_1, .., J_n$ and $\mathcal{T}$ is the set of time points t corresponding to the beginning or to the end of a time window of a job. There is an edge between J_i and t if and only if the job can start at time t. Two more vertices σ, ε are added to the network. The source σ is connected to all jobs and each time vertex is connected to the sink ε. Finally all capacities are 1 except on the edges connecting a time vertex t to ε where the capacity equals the distance between t and the next time point in $\mathcal{T}$ (if no such point exists the capacity is 0). It is easy to see that there is a feasible schedule if and only if there is a flow of capacity n.

3.2 Single Time Window

This situation corresponds to the well known problem $1|r_i, p_i = p|$feasibility and it can be solved in time $O(n \log n)$, by Garey, Johnson, Simons and Tarjan Algorithm [9]. As this problem plays a crucial role for the runway problem (espeacially in Section 6), we briefly describe a simple variant of Garey et al. Algorithm. XXX Konstantin

3.3 Two Tight Windows per Job

We assume that there are two time windows per job and that they are "tight", *i.e.*, their size is exactly p. So we basically have to decide which one to pick. This assignment problem can be solved by a reduction to the 2-Satisfiability problem. To every job i associate a boolean variable x_i which is true if i is scheduled in its first time window and false if it is scheduled in its second time window. For every pair of intersecting time windows with associated literals a, b there will be a clause $\bar{a} \vee \bar{b}$, which translates the condition that no two jobs can intersect. The resulting 2-SAT problem can be solved in polynomial time [11].

3.4 When $p = l$ and T is fixed (Mono-Pattern)

We describe a dynamic programming algorithm running in $O(n^{T+1}s)$ that solves the runway problem when $p = l$.

Let us define the *type* of a job i as $r_i \bmod T$. Now consider two jobs i, j with the same type and assume that $r_i \leq r_j$. It is easy to see that, schedules in which i is scheduled before j are dominant (i.e., if there exists a feasible schedule, there is one in which i is scheduled before j). Indeed, if j is scheduled before i, then we can just exchange the starting times of jobs. Note that according to the above remark, we can restrict our search for a feasible schedule to those in which jobs of the same type are scheduled in order of their release dates.

To simplify the presentation of the algorithm, we assume that jobs are sorted in non-decreasing order of release date. Suppose there are n_a jobs of type $a \in \{0, \ldots, T-1\}$, so $n = \sum_a n_a$. We identify each job i by a couple (a, b) where a is the type of the job, and b is the number of jobs $j \geq i$ of type a. We define the boolean variable $S(t, k_0, \ldots, k_{T-1})$ as follows: It is true if and only if there is a schedule, idle before time t, that contains all jobs (a, b) with $b \in \{1, \ldots, k_a\}$. Given this definition, $S(t, 0, \ldots, 0) =$ true. And we look for $S(\min_i r_i, n_0, \ldots, n_{T-1})$.

Now, let $a = t \bmod T$. If the starting time of the last window of job (a, k_a) is earlier than t then $S(t, k_0, \ldots, k_{T-1}) =$ false, since the job (a, k_a) cannot be scheduled in a schedule which is idle before t. Otherwise $S(t, k_0, \ldots, k_a, \ldots, k_{T-1})$

equals

$$S(t+1, k_0, \ldots, k_a, \ldots, k_{T-1}) \vee S(t+p, k_0, \ldots, k_a - 1, \ldots, k_{T-1})$$

because any schedule idle before t is either idle before $t+1$ as well or contains a job of type $t \bmod T$ that starts at t.

Note that there are at most ns relevant values for t, since in the previous formula, we can replace $t+1$, by the starting time of the next window. So there are $O(n^{T+1}s)$ variables, and they can be computed in constant time (provided we process them in the right order). Moreover, if $s \geq N$, the problem can be solved in polynomial time with greedy algorithm even without the condition $p = l$.

3.5 When sT/p is fixed (Mono-Pattern)

We describe a dynamic algorithm, running in $O(n^2 s^3 T^2 p^{-2} 2^{2sT/p})$. So the problem is polynomial if sT/p is fixed.

In this section we are only interested in *dominating* partial schedules which minimize the completion time among all schedules on the same job set. Moreover we want this property to be preserved when removing recursively the last scheduled job. Such schedules have in particular the property of being *left-shifted*, meaning every job is scheduled either at the beginning of one of its time windows or right at the end of another job. In such a schedule the finishing time points of jobs can be taken from the set $\Theta = \{r_{ij} + ap : i, a \in \{1, \ldots, n\}, j \in \{1, \ldots, s\}\} \cup \{t_0\}$ to which the time $t_0 = \min r_{ij}$ is added for convenience.

Before running the dynamic algorithm of Figure 3 we verify the very basic necessary condition: for every time interval $[a, b]$, the number of jobs which have all their windows contained in $[a, b]$ ($a \leq r_i, r_i + (s-1)T + l \leq b$) does not exceed $(b-a)/p$. Clearly these jobs must be scheduled in this interval and at most $(b-a)/p$ jobs fit.

It follows from the construction that $\mathcal{S}_t$ contain all *dominating* partial schedules, each containing all jobs due at t. In particular $\mathcal{S}_{t_{\max}}$ contains all feasible schedules for the given runway problem, where $t_{\max}$ is the maximal time from Θ.

In order to evaluate the time spent to compute this set, we bound the number of schedules in each set $\mathcal{S}_t$. At time t there are 3 types of jobs. Jobs which are due at t, jobs which are not yet available at $t-p$ and the remaining jobs. The later satisfy $r_i \leq t-p$ and $r_i + (s-1)T + l > t$. By the necessary condition checked earlier for say $a = t - sT$ and $b = t + sT$ there are at most $2sT/p$ remaining jobs. Therefore $|\mathcal{S}_t| \leq 2^{2sT/p}$. We have $|T| = O(sn^2)$. In each of the $|T|$ iterations we need to explore at most $2^{2sT/p}$ schedules and at most $2sT/p$ jobs. We represent schedules by a table of n scheduling times with a special value for yet unscheduled jobs. Schedule sets are represented by search tree based dictionaries, mapping job

Dynamic Algorithm 2
$\mathcal{S}_{t_0} = \{\}$.
For all $t \in \Theta \backslash \{t_0\}$ in increasing order
  Let $t' \in \Theta$ be the latest time point such that $t' < t$.
  Let $t'' \in \Theta$ be the latest time point such that $t' \leq t - p$.
  Let $D = \{i : r_i + (s-1)T + l \leq t\}$ be the set of jobs due at t.
  Let $A = \{i : r_i \leq t - p\}$ be the set of jobs available at $t - p$.
  Set $\mathcal{S}_t = \{S \in \mathcal{S}_{t'} :$ all jobs in D are scheduled in $S\}$.
  For all $S \in \mathcal{S}_{t''}$
    For all jobs $i \in A$ not yet scheduled in S
      If $S + i$ contains all jobs in D
        and there is no schedule in $\mathcal{S}_t$ with the same job set as $S + i$
      then add $(S + i)$ to $\mathcal{S}_t$.

Figure 3: Dynamic algorithm for sT/p fixed.

sets to schedules. Therefore adding a new schedule or checking if there is already a schedule on the same job set has logarithmic cost, which is $O(sT/p)$ in our case. Therefore total running time is $O(n^2 s^3 T^2 p^{-2} 2^{2sT/p})$.

3.6 When Time Windows are Well Ordered (Mono-Pattern)

When $\forall i, \max r_i + l \leq \min r_i + T$ then the problem can be solved by Linear Programming. As this is a special case of a more general situation, the LP formulation is described in Section 5 and Theorem 2.

We first restrict our study to the Mono-Pattern problem and we show that, due to structural properties of the problem, we have "small" MIP formulation of the problem (Section 5).

4 Bayen et al. MIP formulation

Bayen, Tomlin, Ye and Zhang have introduced a MIP formulation for maximizing the minimum time elapsed between two consecutive landings [6]. As they work on the minimization problem, the minimum ellapsed time between two consecutive landings is a variable P. Each job i is associated with a starting time variable t_i. Moreover, two sets of integer variables are used:

- x_{iu} variables are binary variables that indicate whether aircraft i is schedule in one of its uth first time windows or not.

- y_{ij} variables are binary sequencing variables that indicate whether aircraft i precedes j or not.

The complete MIP is then as follows:

$$\min P$$
$$\begin{cases} t_i \geq r_{i1} & 1 \leq i \leq n \\ t_i \leq d_{is_i} & 1 \leq i \leq n \\ t_i \geq r_{iu} - M x_{i\,u-1} & 1 \leq i \leq n, 2 \leq u \leq s_i \\ t_i \leq d_{iu} + M(1 - d_{i\,u}) & 1 \leq i \leq n, 1 \leq u \leq s_i - 1 \\ t_i - t_j \geq P - M' y_{ij} & 1 \leq j < i \leq n \\ t_i - t_j \leq M'(1 - y_{ij}) - P & 1 \leq j < i \leq n \\ x_{iu} \in \{0,1\} & 1 \leq i \leq n, 1 \leq u \leq s_i - 1 \\ y_{ij} \in \{0,1\} & 1 \leq i \leq n, 1 \leq i \leq n \end{cases}$$

where M and M' are two large constrant values chosen as follows: Such large values usually lead to poor LP relaxation and hence, as we will see in our experiments, to poor performance of the MIP.

$$\begin{aligned} M &= \max_i \max(r_{is_i} - r_{i1}, d_{is_i} - d_{i1}) \\ M' &= 2(\max_i d_{is_i} - \min_i r_{i1}) \end{aligned}$$

We have tightened this formulation by adding the cuts $x_{iu} \geq x_{i\,u-1}$ and $y_{ij} + y_{ji} = 1$.

5 A Small MIP for the Mono-Pattern Problem

We come back to the *decision* variant of the mono-pattern problem. So now, the minimal time ellapsed between two consecutive landings is fixed (p) and we present a MIP formulation for the Mono-Pattern problem. To every j-th time window of every job i we associate a binary variable $x_{ij} \in \{0,1\}$, stating that job i is processed in this time window or not. In every interval $[a,b]$ at most $\lfloor (b-a)/p \rfloor$ jobs can be strictly contained. Then the following inequalities must hold:

$$\begin{cases} \sum_{u=1}^{s_i} x_{iu} = 1 & 1 \leq i \leq n \\ \sum_{[r_{iu}, d_{iu}] \subseteq [a,b]} x_{iu} \leq \lfloor \frac{b-a}{p} \rfloor & 1 \leq i \leq n, a \in \{r_{jv}\}, b \in \{d_{jv}\}, a < b \\ x_{iu} \in \{0,1\} & 1 \leq i \leq n, 1 \leq u \leq s_{iu} \end{cases} \quad (1)$$

Note that (1) is a MIP with no objective function.

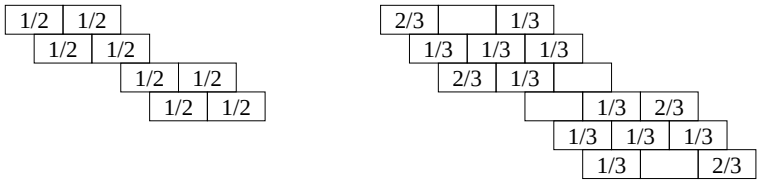

Figure 4: Two instances of the runway problem with $p = l = T = 2$ which have only fractional solutions.

Theorem 1. *There is a solution to (1) if and only if there is a feasible schedule.*

Proof. The sufficient condition is obvious. To prove the necessary condition, we first recall a well-known result of single machine scheduling theory $1|r_j, d_j, pmtn|-$: Consider a set of jobs with arbitrary processing times (p_i), release dates (r_i) and deadlines (d_i). There is a feasible preemptive schedule if and only if over any time interval $[t_1, t_2]$, the sum of the processing times of the jobs i such that $t_1 \leq r_i$ and $d_i \leq t_2$ is not greater that $t_2 - t_1$ (see for instance [8]).

Now, assume we have a solution to the MIP and, for all i, let $j(i)$ denote the index such that $x_{ij(i)} = 1$. If we define $r_i = r_{ij(i)}$ and $d_i = r_{ij(i)} + l$ then we know that over any time interval $[t_1, t_2]$, the sum of the processing times of the jobs that have to be processed in this time interval is not greater than $t_2 - t_1$. Thanks to the previous remark, this directly ensures that we have a feasible preemptive schedule in which jobs i are scheduled between $r_{ij(i)}$ and $d_i = r_{ij(i)} + l$.

The preemptive schedule can be easily transformed into a non-preemptive one. Indeed, it is well-known that if there is a preemptive feasible schedule, then the preemptive schedule associated to the dispatching rule "Earliest Deadline First" is feasible. Recall that in our case, all time windows have the same length s. Hence when, according to the "Earliest Deadline First" rule, a job starts to be processed, it is never interrupted. Hence we have a non-preemptive schedule. □

After having run initial experiments using the MIP formulation, it happened that the gap between the LP relaxation and the optimal solution was always 0. It took us some time to build an instance with a non null gap. In Figure 4(left) we give an instance for $s = 2$ with a fractional solution to the LP for which there is no integer solution: job 1 must be scheduled in its first time window, otherwise there would be no space left for job 2. For similar reason job 4 must be scheduled in its second time window. And the remaining jobs have only a single time window left each, which intersect. For completeness we also show an instance which has fractional but no integer solution for $s = 3$, since the subcase $p = l$ and $s = 2$ is polynomial solvable anyway.

There is however a case for which the LP formulation gives a sufficient condition for the existence of a schedule. For convenience, we say that the j-th window of a job has *rank* j.

Theorem 2. *If* $\max r_i + l \leq \min r_i + T$ *(only windows of the same rank can intersect) then there is a feasible schedule if and only if the LP relaxation of the MIP has a solution.*

Proof. Assume that the LP relaxation of the MIP has a solution $\tilde{x}$. We will show that there is a feasible schedule. (The other direction follows from Theorem 1.) In fact we will show that there is an integer solution $\bar{x}$.

Assume the jobs are ordered according to release times. Now set $\bar{x}_{ij} = 1$ if $j \equiv i \pmod{s}$ and $\bar{x}_{ij} = 0$ otherwise. We claim that this is a solution to the MIP and that consequently, we have a feasible schedule.

First we observe a property of the linear program. By hypothesis j-th time window of a job does not intersect with the $j+1$-th time window of another job. Therefore inequalities depending on a beginning of a j-th time window and on b end of a $j+1$-th time window can be obtained by adding the inequalities associated to (a, b') and (a', b) where b' is the end of the latest j-th time window and a' the beginning of the first $j+1$-th time window. Therefore it is sufficient to keep in the linear program, inequalities for (a, b) where a, b correspond to time windows of the same rank. To show that $\bar{x}$ is a solution we need to show that the inequality

$$\sum_{[r_{ij}, r_{ij}+l] \subseteq [a+(j-1)T, b+(j-1)T]} \bar{x}_{ij} \leq \left\lfloor \frac{b-a}{p} \right\rfloor, \tag{2}$$

holds for arbitrary $j \in \{1, \ldots, s\}$ and $a, b \in [\min r_i, \max r_i + T]$.

Let c be the number of jobs i with $a \leq r_i \leq b - l$. Since $\tilde{x}$ is a solution we have

$$\sum_{[r_{ij}, r_{ij}+l] \subseteq [a+(j-1)T, b+(j-1)T]} \tilde{x}_{ij} \leq \left\lfloor \frac{b-a}{p} \right\rfloor.$$

Summing over all window ranks $j \in \{0, \ldots, s-1\}$ we get

$$c \leq s \left\lfloor \frac{b-a}{p} \right\rfloor.$$

Therefore

$$\frac{c}{s} \leq \left\lfloor \frac{b-a}{p} \right\rfloor.$$

Since the right hand side is an integer we have also

$$\left\lceil \frac{c}{s} \right\rceil \leq \left\lfloor \frac{b-a}{p} \right\rfloor.$$

But by construction of $\bar{x}$, we have

$$\sum_{[r_{ij},r_{ij}+l]\subseteq[a+(j-1)T,b+(j-1)T]} \bar{x}_{ij} \leq \left\lceil \frac{c}{s} \right\rceil$$

from which inequality (2) follows. □

6 A Branch and Cut for the General Problem

Now we study the general problem and as for the mono-pattern problem, we can rely on MIP (1). Unfortunately, Theorem 1 does not hold any more. However, the last part of the proof only does not hold and we still have a weaker version of this thorem:

Theorem 3. *There is a solution to (1) if and only if there is a feasible preemptive schedule.*

Of course all constraints of MIP (1) are still valid in the non preemptive case and it is "likely" that the assignement of jobs to windows also leads to a non pre-mmptive schedule. To test this, we can apply the $O(n \log n)$ algorithm of Garey, Johnson, Simons and Tarjan [9].

- If there is non-preemptive schedule then we have found a solution to the runway problem.
- Otherwise, we know that the assignement found by MIP (1) does not lead to a feasible non-preemptive schedule. Hence, we can add an "iterative" cut to the MIP stating that the sum of the x variables that were equal to 1 in the previous solution cannot be greater than $n - 1$. We then solve the new MIP and iterate the same process.

This branch and cut mechanism terminates since, each time a cut is added, a yet unexplored assignement will be generated at the next iteration. Of course the total number of added cuts can be very large. To make this mechanism work, several ingredients have to be added.

- Preprocessing cuts can be used to tighten the inital MIP formulation (1).
- The generation of iterative cuts can be improved to reduce the total number of iterations.

6.1 Preprocessing Cuts

We describe two sets of cuts: Disjunctive Cuts [13, 15] and Energetic Cuts. Note that in both cases, these disjunctive cuts are only valid in the non-preemptive case. We have tested several other kind of cuts. Some of them were leading to a very limited number of iterations but the size of the MIP was increasing so much that the time required to solve it was too large. In the following we describe two sets of $O(n^2)$ cuts.

6.1.1 Disjunctive Cuts

These cuts prevent the simultaneous assignment of a pair of jobs i, j to some time windows $[r_{iu}, d_{iu})$, $[r_{jv}, d_{jv})$, if the situation is "infeasible". The feasibility test is performed as follows:

- Relax the time windows of jobs $k \notin \{i, j\}$ to one single time winow $[r_{k1}, d_{kn_k})$.
- Tighten the time windows of i, j to $[r_{iu}, d_{iu})$, $[r_{jv}, d_{jv})$.
- As in the resulting instance jobs have a single time window, the feasibility test can be achieved thanks to Garey et al. algorithm

As there are $O(n^2)$ pair of jobs to consider this leads to an $O(n^3 \log n)$ (a bit more in our naive implementation) algorithm to build all the disjunctive cuts.

6.1.2 Energetic Cuts

The notion of "energy" has been originally introduced for constraint propagation techniques (see [1] for instance). Consider a time interval $[t_1, t_2)$ with $t_1 + p \leq t_2$. Preemtive constraints of MIP (1) ensure that the number of jobs scheduled in the interval does not exceed $\nu = \left\lfloor \frac{t_2 - t_1}{p} \right\rfloor$. Now assume that we have a job i and a time window u such that $r_{iu} + (\nu + 1)p > t_2$ and $t_1 + (\nu + 1)p > d_{iu}$ then it is impossible to schedule non preemptively the job i if we have ν jobs in the interval $[t_1, t_2)$. This leads to the cut:

$$x_{iu} + \sum_{j,v : t_1 \leq r_{jv} \leq d_{jv} \leq t_2} x_{jv} \leq \nu$$

6.2 Iterative Cuts

Recall that an iterative cut is generated when Garey, Johnson, Simons and Tarjan Algorithm [9] discards an assignement proposed by the MIP (*i.e.*, the MIP has build a preemptive schedule but it cannot be changed into a non preemptive one).

A valid iterative cut is a cut which guarantees that the same assignment will not be generated in the subsequent iterations. Our intuition is that "small" cuts, *i.e.*, cuts with few variables, are a priori better than large cuts.

So, we examine the current solution of the MIP and we compute a minimal set of jobs (as assigned in the solution) that lead to an unfeasible non-preemptive scheduling situation. As we do not want to try an exponential number of sets, we restrict our search to a quadratic number of sets. Each set is defined by two integer t_1, t_2 and is made of jobs i whose assigned window is included in $[t_1, t_2]$. More precisely, we use the following algorithm.

- Iterate over all possible release dates for t_1 and all possible deadlines for t_2.
- Test with Garey, Johnson, Simons and Tarjan Algorithm if the jobs assigned in the time windows contained in $[t_1, t_2)$ can be scheduled, if not, consider this set of jobs as a candidate cut.
- Find a minimal cut among the candidates.

7 Experiments

To compare our approach with Bayen et al., we have led our experiments on the maximization of the minimum ellapsed time between aircrafts. All our algorithms have been designed to work on the decision variant of this problem so, we just apply a dichotomic search on the objective value and solve successive variants of the decision problem: We starty by by computing an initial upper bound

$$p_{\max} = \left\lfloor \frac{\max_i d_{in_i} - \min_i r_{i1}}{n-1} \right\rfloor$$

and an initial lower bound $p_{\min} = 0$ for the objective. Then

1. Set $p = \left\lfloor \frac{p_{\min} + p_{\max}}{2} \right\rfloor$.
2. Solve the descision variant of the runway problem with processing time p, *i.e.*, determine a solution with a "minmum ellapsed time between aircrafts" lower than or equal to p or prove that no such solution exists. If a solution is found, set $p_{\min}$ to the value of minmum ellapsed time between aircrafts in the solution; otherwise, set $p_{\max}$ to p.
3. Iterate steps 1 and 2 until $p_{\max} = p_{\min}$.

Two sets of instances have been generated (all instances are available on the web XXXX).

- The first set of instance corresponds to the mono-pattern problem. XXX Konstantin tu nous expliques en 5 lignes.
- The second set of instances corresponds to the general problem. Instances with 15, 30, 45, 60, 75 and 90 jobs have been ranomly generated (according to three parameters π, ι and λ) as follows:
  - The number of intervals for a given job is a random number taken betwen 1 and 5.
  - The length of an interval is a random number in $\pi\iota$.
  - The distance between two consecutive intervals is a random number in $\pi\lambda$.

  For each value of n, the following combination of parameters have been tested $\pi \in \{75, 150\}$, $\iota \in \{\pi, 6\pi, 11\pi, 16\pi\}$, $\lambda \in \{\pi, 6\pi, 11\pi, 16\pi\}$. This leads to 64 instances for each size.

We have used Ilog Cplex 9.0.3 to solve the MIP and all tests have been performed on a 1.4 GHz PC running Windows XP.

7.1 Mono-Pattern Problems

We first compared the MIP proposed by Bayen et al. (MIP-BY in the following) with our MIP (MIP-1 in the following). In the first case, a single MIP is solved while in the second case we solve several MIPs (as many as the numbe of iterations of the dichotomizing algorithm). In both case the search has been stoped afer 600.00 seconds of CPU time. Our algorithm solves 95 instances out of 98 while MIP-BY can solve 23 instances only. Over the instances that could be solved by both algorithms, ours found the optimal solution in an average CPU time of 26.3 seconds and less than 1 node while MIP-BY required 55.1 seconds and 57018 nodes.

As stated in Section 4, the initial formulation of Bayen et al. can be tightened by adding the cuts like $x_{iu} \geq x_{i\,u-1}$. This allows us to slightly improve the behaviour of the algorithm since 6 more instances are solved.

7.2 General Problem

Several variants of the Branch and cut have been tested.

- Either we apply the basic iterative cuts (BAS-CUT) or we use the mechanism computing small cuts (SMALL-CUT).

- Either we do not apply any preprocessing cuts (NO-PP) or we apply the disjunctive cuts (DISJ-PP) or we apply the disjunctive cuts plus the energetic cuts (ENERGY-PP).

This leads to six variants “NO-PP, BAS-CUT”, “DISJ-PP, BAS-CUT”, “ENERGY-PP, BAS-CUT”, “NO-PP, SMALL-CUT”, “DISJ-PP, SMALL-CUT”, “ENERGY-PP, SMALL-CUT” plus the the MIP formulation of Bayen et al. Figure 5 reports the number of instances solved withing a given amount of cpu time for each of these 7 programs (search was stoped after 100.00s). The testbed was made of the 192 instances with less than 45 jobs.

These results show that Bayen et al. formulation performs well in the first few seconds of CPU time (more instances are solved by this formulation than for any other). Howevever, our Branch and Cut compares well as soon as some more time is given to solve instances. All versions except the most basic one clearly outperform Bayen et al. formulation. Within the time limit, it can sole 108 instances while “NO-PP, BAS-CUT”, “DISJ-PP, BAS-CUT”, “ENERGY-PP, BAS-CUT”, “NO-PP, SMALL-CUT”, “DISJ-PP, SMALL-CUT”, “ENERGY-PP, SMALL-CUT” solve respectivel 108, 105, 181, 182, 134, 188 and 189 instances. Note that the 3 instances that could not be solved by “ENERGY-PP, SMALL-CUT” are solved within 150.00 seconds.

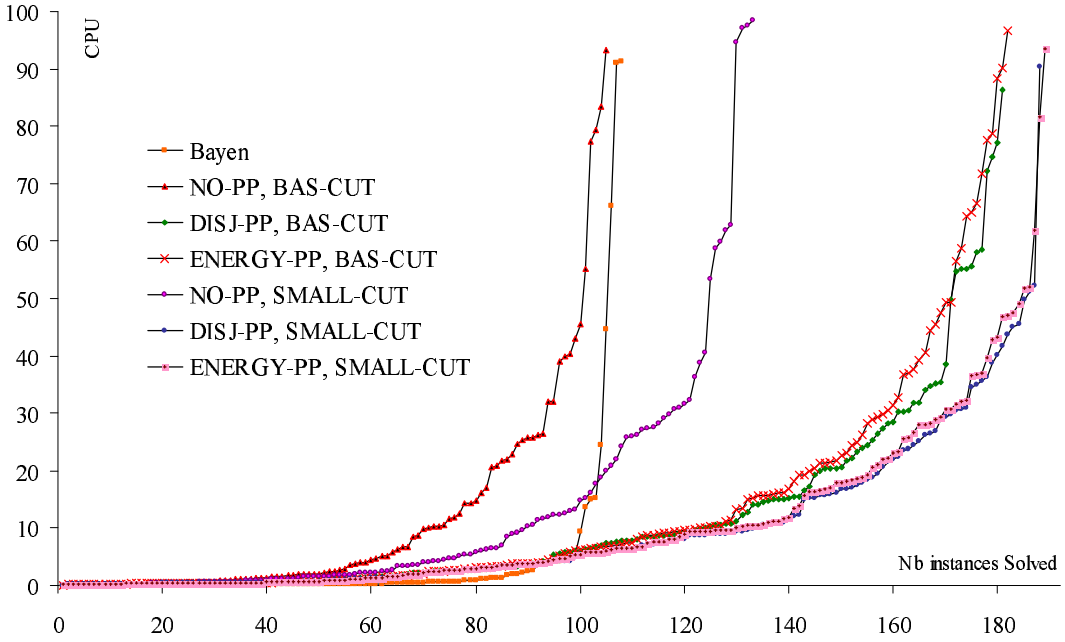

Figure 5: Number of 15, 30 &45 job instances solved within a CPU limit

Surprisingly our MIPS seem to be easy to solve since on the average 10 nodes are developped by Cplex Branch and Bound procedure (much much more for Bayen et al. formulation). Another good news is that the total number of iterations (*i.e.*, the total number of iterative cuts that we have added) is kept low (5 on the average with a maximum of 104). Following these experiments, it seems clear that two ingredients play a crucial role for the branch and cut:

- The disjunctive cuts are extremly powerfull and drastuically reduce the total number of iterations. Hence they more or less guarantee that preemptive solutions are likely to be transformed in non-preemptive ones.
- The choice of small cuts is also important and allows to reduce the number of iterations.
- The usefullness of the Ebnergetic cuts is not very clear although it does not cost much to use them.

Following these results we decided to test the best version of our Branch and Cut "ENERGY-PP, SMALL-CUT" on instances with 60, 75 and 90 jobs. 62 instances with 60 jobs, 58 instances with 75 jobs and 36 instances with 90 jobs are solved within 10 mn.

8 Conclusion

Some of the programs described in this note are available at http://www.lri.fr/~durr/runway/.

To conclude, we would like to mention that, up to our knowledge, the general problem is still open.

Although we conjecture that the problem is NP Hard, we have not been able to prove it. However, some very simple generalizations are NP-Hard.

Acknowledgement

We would like to thank Nodari Vakhania and Baruch Schieber for fruitful discussions. The second and third authors are supported by NSF/CNRS grant 17171. The third author is also supported by the EU 5th framework programs QAIP IST-1999-11234 and by CNRS/STIC 01N80/0502 and 01N80/0607 grants.

References

[1] Ph. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-based Scheduling*, volume 39 of *ISOR*. Kluwer Academic Publishers, 2001.

[2] Ph. 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(1999), 245–252.

[3] N. Barnier and P. Brisset, Slot Allocation in Air Traffic Flow Management http://www.recherche.enac.fr/opti.

[4] A. M. Bayen and C. J. Tomlin, Real-time discrete control law synthesis for hybrid systems using MILP: Application to congested airspace, Proceedings of the American Control Conference, 2003.

[5] A. M. Bayen, C. J. Tomlin, Y. Ye, J. Zhang, An Apporixmation Algorithm for Scheduling Aircraft with Holding Time, To appear in the Proceeding of the 43rd IEEE Conference on Decision and Control, 2004.

[6] A. M. Bayen, C. J. Tomlin, Y. Ye, J. Zhang, MILP formulation and polynomial time algorithm for an aircraft scheduling problem, cherokee.stanford.edu/~bayen/publications.html.

[7] J. Carlier. The One-Machine Sequencing Problem. *European Journal of Operational Research*, 11:42–47, 1982.

[8] J. Carlier. Problèmes d'Ordonnancement à Contraintes de Ressources : Algorithmes et Complexité. Thèse de Doctorat dEtat, Université Paris VI, 1984.

[9] M.R. Garey, D.S. Johnson, B.B. Simons, and R.E. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM Journal on Computing*, **10**(2), 256–269, 1981.

[10] J. Hooker. A hybrid method for planning and scheduling, April 2004, Tenth International Conference on Principles and Practice of Constraint Programming, Toronto, Canada, 2004.

[11] J.M. Keil, On the complexity of scheduling tasks with discrete starting times, *Operation Research Letters*, **12**, 293–295, (1992).

[12] B. Simons. A fast algorithm for single processor scheduling. 19th Annual Symposium on the Foundations of Computer Science, pages 246–252, October 1978.

[13] R. Sadykov and L. Wolsey. Integer programming and constraint programming in solving a multi-machine assignment scheduling problem with deadlines and release dates. Discussion paper 2003/81, Center for operations research and econometrics, Catholic University of Louvain, 2003.

[14] F.C.R. Spieksma and Y. Crama, The complexity of scheduling short tasks with few starting times, Reports in operation research and system theory M92-06, Maastricht University.

[15] A. Vazacopoulos and N. Verma. Hybrid LP-CP techniques to solve a Multi-Machine Assignment and Scheduling Problem. Forthcoming in "Supply Chain Optimization", Editors: Joseph Geunes and Panos Pardalos, Dec 2004.