Optimizing Airspace Closure with Respect to Politicians' Egos[1]

Irina Kostitsyna[1,*]

Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands

Maarten Löffler[2]

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Valentin Polishchuk[3]

Communications and Transport Systems, ITN, Linköping University, Sweden

Abstract

When a president is landing at a busy airport, the airspace around the airport closes for commercial traffic. We show how to schedule the presidential squadron so as to minimize its impact on scheduled civilian flights; to obtain an efficient solution we use a "rainbow" algorithm recoloring aircraft on the fly as they are stored in a special type of forest. We also give a data structure to answer the following query efficiently: Given the president's ego (the requested duration of airspace closure), when would be the optimal time to close the airspace? Finally, we study the dual problem: Given the time when the airspace closure must start, what is the longest ego that can be tolerated without sacrificing the general traffic? We solve the problem by drawing a Christmas tree in a delay diagram; the tree allows one to solve also the query version of the problem.

Keywords: scheduling; data structure; algorithms
2010 MSC: 68M20, 68P05, 68W40

1. Introduction

Airspace closure due to military activities is a pain for civilian air traffic controllers (ATCOs), pilots, airlines and other stakeholders; the issue is especially notorious in countries with heavy military control of the skies (such as China). Military flight operations range from strike and defense missions to drills to humanitarian airdrops. The missions are impossible to reschedule, and military ATCOs are entitled to ceasing airspace from civilian use at any time when the traffic could conflict with the mission aircraft. Drills are better in this regard because they are planned in advance, but nevertheless airspace closure due to drills harms commercial airlines. On the contrary, humanitarian aid delivery typically has little effect on general air traffic—not the least due to the fact that the aid is often delivered to places far from mainstream airports.

There is one activity involving military air force, however, whose scheduling most certainly could have been done wiser than it is done today: air transfer of VIPs (presidents and other high-ranked politicians). We trust that planners of such activities are instructed by their superiors (the VIPs at hand) to take civilian needs

[1]Preliminary version appeared at the 7th International Conference on FUN WITH ALGORITHMS [1].

[*]Corresponding author
Email addresses: i.kostitsyna@tue.nl (Irina Kostitsyna), m.loffler@uu.nl (Maarten Löffler), valentin.polishchuk@liu.se (Valentin Polishchuk)

[1]Irina Kostitsyna is supported in part by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.106.
[2]Maarten Löffer is supported by the Netherlands Organisation for Scientific Research (NWO) under grant no. 639.021.123.
[3]Valentin Polishchuk's work is supported by grant 2014-03476 from the Sweden's innovation agency VINNOVA.

into account when planning the flights—all VIPs are conscientious citizens putting needs of the people above their personal comfort. Unfortunately, no matter how hard the planners strive to follow the instructions, the civilians do get annoyed with delays caused by VIP flights. For instance, the recent visit of the US President to Sweden disrupted air traffic to the Stockholm area and gave rise to heated discussions among professionals in the Malmö Air Traffic Control Center (in the south of Sweden) about possible measures that could have been taken to diminish the disruption [2]. Apparently, the only reason preventing the planners of VIP flights from requesting airspace closure with minimum impact on scheduled traffic is the absence of efficient algorithms for computing the optimum (we were not able to find prior work on the issue).

In this paper we set out to alleviate the difficulty by providing algorithms for deciding the optimal airspace closure timing. Employing our solutions will make the general public happier about VIPs, which will eventually pay back to politicians at future elections.

Background. Steady growth of air transportation poses political, industrial, technological and other challenges to mankind; these challenges are addressed by the US Next Generation Air Transportation System (NextGen) and the EU Single European Sky Air Traffic Management Research (SESAR) Joint Undertaking. Both NextGen and SESAR view Flexible Use of Airspace (FUA) as one of the tools for provision of safe and efficient air transport in the future, and improving cooperation between military and civilian traffic is an important part of FUA; see, *e.g.*, [3].

For every aircraft an optimal flight plan exists (it can be fuel-optimal or time-optimal or optimal according to another objective) which includes both the altitude profile and the speed at every point along the path. Whether the aircraft is able to execute such a plan depends heavily on the other aircraft around. In the uncongested enroute portion of the flight, aircraft with similar headings can generally "overtake" each other; to quote Director General of Luftfartsverket (Swedish air navigation service provider) [4], the uncontrolled oceanic airspace witnesses "air race over the Atlantic" on a daily basis. On the contrary, in the vicinity of an airport, the arrival manager sequences aircraft "ducks-in-a-row" to the approach; here, faster aircraft (those whose desired speed is larger) must slow down in order to maintain separation from the preceding slower plane. This latter scenario is the one considered in this paper.

1.1. Model

Assume that the approach to an airport is a "single-lane road" of length 1. The approach does not have to be a straight-line segment (in fact, in the real world, approaches to many airports are curved); the important thing is that it is a one-dimensional curve. Aircraft enter the approach at times $t_1, \ldots, t_n$ (known from the schedules or flight plans) and have desired speeds $v_1, \ldots, v_n$ (known from communication between pilots and ATCOs or from automated flight management systems); n is the number of aircraft. For simplicity assume all t_is and all v_is are distinct. In absence of other planes, aircraft i would traverse the approach uniformly at speed v_i, arriving at the airport at time $\tau_i = t_i + 1/v_i$; this is an oversimplification, but our solutions can be modified (possibly, at the expense of increased runtimes) to work with arbitrary desired speed profiles (we leave the details for future work). Since overtaking is not allowed on the approach, any aircraft must slow down to the speed of the preceding plane as soon as the aircraft catches up with the plane. In other words, we assume that aircraft have 0 length and can follow each other without gaps; this is also an oversimplification, but again, our algorithms can work under the requirement of minimum safe separation distances between aircraft as well (in reality, the minimum miles-in-trail restriction is not uniform—it depends on the types of aircraft, with light aircraft having to stay farther behind a heavier one to avoid wake vortices). That is, in the tx-plane, the desired trajectory of aircraft i is the segment s_i going from $(t_i, 0)$ to $(\tau_i, 1)$. However, if the aircraft catches up with a slower one, they start moving together at the slower speed; thus the actual aircraft location on the approach is a concave piecewise-linear function of time. In the tx-plane the trajectories get merged into trees corresponding to *platoons* of aircraft (Figure 1, left), where a platoon is a set of aircraft that land together. We call the first (slowest) aircraft in a platoon the *head aircraft*.

Consider platoon p consisting of several aircraft $\{1, 2, \ldots, j\}$, sorted in order they enter the approach. Aircraft 1 is the head aircraft, and the rest aircraft $2, \ldots, j$ eventually catch up and follow 1 at its speed. Consider some aircraft u in p (where $1 < u < j$), and let w (where $u < w \le j$) be the next aircraft in p that is slower than u, *i.e.*, aircraft $u+1, \ldots, w-1$ are all faster than u. Then aircraft $u, u+1, \ldots, w-1$ form a

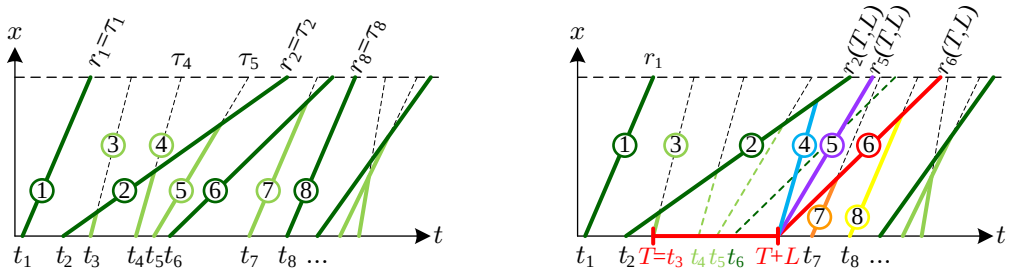


Figure 1: Left: The desired trajectories are dashed, the actual trajectories are green; the green segment starting at $(t_i, 0)$ is the part that aircraft i travels with its desired speed v_i. Trees correspond to platoons of aircraft landing at the same time; dark green aircraft 1, 2, 6, 8, etc., are platoon heads. Right: The trajectories change due to the closure (the aircraft are colored according to our algorithm in Section 2).

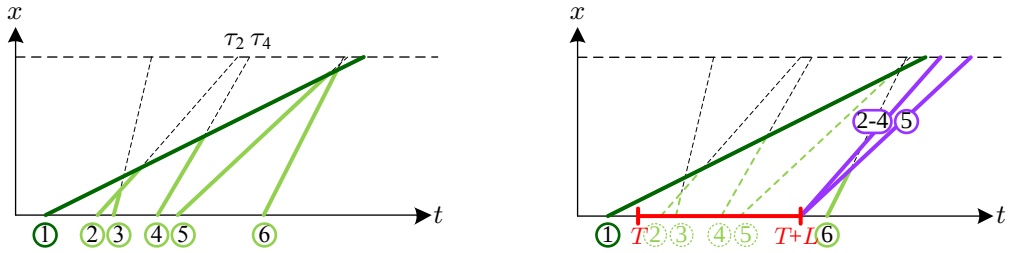


Figure 2: Left: A platoon consisting of six aircraft $\{1, 2, \ldots, 6\}$. Here, for example, aircraft 2, 3 and 4 form a *subplatoon*. Notice that even though $\tau_2 < \tau_4$ (trajectories of aircraft 2 and 4 would not intersect in the absence of other aircraft), we still consider $\{2, 3, 4\}$ to be a subplatoon. Other subplatoons are $\{3\}$, $\{4\}$, $\{5, 6\}$, and $\{6\}$. Right: Airspace closure interval $(T, T+L)$ delays all aircraft with approach entry times greater than T. A subplatoon $\{2, 3, 4\}$ enters the approach at time $T+L$, all aircraft in the subplatoon move together with the speed of the heading aircraft. This illustrates, why we chose such a definition of a subplatoon (more details in Section 2.4).

subplatoon (Figure 2). If there is no such aircraft w that is slower than u, where $u < w \leq j$, then aircraft $u, u+1, \ldots, j$ form a subplatoon.

Let r_i denote the actual time when aircraft i lands ($r_i = \tau_i$ for platoon heads); the points $(r_i, 1)$ are roots of the trajectory trees. The trees can be built from the segments $s_1, \ldots, s_n$ by modifying the Bentley–Ottmann sweep for segment intersection [5, Chapter 7]: just remove the faster segment on any intersection. Since the total complexity of the trees is linear (every segment is removed at most once), the sweep completes in $O(n \log n)$ time. We remark that our algorithms do not need to construct the trees in the tx-plane exactly; we will store only the combinatorial structure of every tree (Section 2).

When a VIP is expected to arrive at the airport, the airspace has to be closed temporarily. All that is known about the VIP is the length L of their ego, which measures for how long the approach will be closed. If the closure starts at time T, then all aircraft with entry times in the open interval $(T, T+L)$ will be put into a holding pattern and will effectively enter the approach at time $T+L$. We assume that the sequence of aircraft does not change at the exit from holding (which is mostly true in the real world). Thus, equivalently, the aircraft entrance is delayed until $T+L$, at which time they all enter in the same order in which they arrived (Figure 1, right). Let $r_i(T, L)$ denote the time when aircraft i will land if the approach is closed during time interval $(T, T+L)$; our cost function is the total sum of the landing times

$$D(T, L) = \sum_{i=1}^{n} r_i(T, L)\,.$$

Minimizing $D(T, L)$ is the same as minimizing the total delay $D(T, L) - \sum_{i=1}^{n} r_i$ caused by the VIP; therefore we will also call $D(T, L)$ the *delay* function. Note that we do not consider aircraft i to be delayed when $r_i(T, L) = r_i > \tau_i$, as it lands at a time that is predefined by the schedule. We only consider an aircraft

delayed if its landing time is affected by the airspace closure (see Section 2.3).

We will consider several cases of the problem: minimize $D(T, L)$ when L is fixed, minimize $D(T, L)$ when T is fixed, and a query version of minimizing $D(T, L)$ for a given L. To avoid trivialities, in the case of fixed L, we assume that there is a lower ($t_{\min} > t_1 - L$) and an upper ($t_{\max} < t_n$) bound on the location of T (otherwise, one would trivially set $T > t_n$ or $T < t_1 - L$).

1.2. Our Contributions

We consider several problems:

Delay minimization (Section 2): *Given L, find T to minimize $D(T, L)$.* To solve the problem in $O(n \log n)$ time we store the combinatorial structure of platoons in a forest which can be updated efficiently as the closure interval slides along the t axis. The forest may be of independent interest, as it can be viewed as a juste milieu between storing platoons as lists and storing the full trajectory trees in the xt-plane (the former does not have enough structure to allow for efficient updates, while the latter contains "too much" structure and therefore is hard to update).

Ego query (Section 3): *The query version of delay minimization.* We give a nearly-quadratic-time algorithm to compute the function $D(T, L)$ for all L and T (a simple example shows that the function can have quadratic complexity). We show that the marginal minimum $f(L) = \min_T D(T, L)$ has $O(n^2\alpha(n))$ complexity, where $\alpha(n)$ is the inverse Ackermann's function, and that it can be built in nearly-quadratic time. Knowing the function $f(L)$ allows one to give a logarithmic-time answer to the delay minimization query "Given L, report T that minimizes $D(T, L)$".

The Harmless President problem (Section 4): *Given T, find the maximum L for which $D(T, L)$ is 0.* This is the dual problem to delay minimization: for the latter we assumed that the VIP would be willing to shift T so as to minimize the delay—this could be an unrealistic assumption since VIPs have tight schedules; the solution to the Harmless President problem helps modest but busy VIPs who cannot reschedule the entry into the airspace but are willing to curb their egos so as to do no harm to the people (of course, the majority of politicians are such, so finding the maximum harm-free L is the most practical problem). The Harmless President problem can be solved in nearly-quadratic time using our results for the Ego query problem: after the function $D(T, L)$ is built we can find, for the given T, the largest L such that (T, L) is in the 0-level (complement of the support) of $D(T, L)$. We show that the 0-level actually has linear complexity and can be built in $O(n \log n)$ time; thus the Harmless President problem can be solved in $O(n \log n)$ time. In addition, using the 0-level, one can solve the query version of the Harmless President problem in logarithmic time per query.

Naturally, our algorithms are applicable to arbitrary scenarios of temporary traffic disruption: train track closure, farmers machinery (or geese, or kids) crossing a rural road, walkway blockage, etc. The algorithms can also potentially be extended to handle (multiple-lane) roads where overtaking is allowed (we leave the details for future work).

1.3. Related Work

Traffic jam formation is a vast research area fueled by the enormous real-life importance of the field. Many studies—ranging from on-site experimental measurements to simulation and modeling to purely theoretical developments—have been performed over the years. Various traffic flow characteristics were explored, most notably the fundamental diagram of the flow (the relationship between flow rate, vehicle speed and density), aiding in understanding flow breakdown, jam formation and other processes occurring in the traffic. Not attempting to survey the huge amount of literature on the subject, we refer to books and surveys [6, 7, 8].

A related domain of active research is motion information gathering: traffic participants (be it cars, aircraft, ships, trains, pedestrians, birds or other animals) are being tracked using mobile phones, GPS navigators, special-purpose devices, etc. One popular form of processing of the gathered data is information summarization, in particular, trajectory clustering; for some recent work see, *e.g.*, [9] and references thereof.

We remark that randomness is inherent to real-world traffic; this is why most models assume that entry times and/or desired speeds are random variables. Nevertheless, in several scenarios (like ours, but also, *e.g.*, for trains) knowing the times and the speeds from schedules (and flight/trip plans) may be a natural assumption. It is also worth noting that the validity of even the most established stochastic models of Kerner and Rehborn [10, 11, 12] has been questioned: [13] gives alternative explanations to empirical data on traffic breakdown. In addition, even papers explicitly dealing with the stochastic component did not assume *all* elements to be probabilistic. For example, one seminal paper on platoon formation [14] had cars arrive uniformly (the car's desired speeds were random variables). Fixing some input parameters allowed Newell [14] to identify the features of the model defining the road capacity.

2. Delay Minimization

In this section we consider the following problem: Given L, find T to minimize $D(T, L)$. First we discuss a simple $O(n^2)$ algorithm to solve the problem. For a more efficient solution we introduce a special data structure and assign colors to aircraft.

2.1. Platoon List

Our first goal is to calculate the landing times r_i for all aircraft given the approach entry times t_i and (ideal) landing times τ_i (recall that $\tau_i = t_i + 1/v_i$ is the time when i would land in solitude). Define the *platoon list* $\mathcal{P}$ to be the list of aircraft platoons sorted by the approach entry times of their head aircraft (which is the same as sorting by landing times).

Lemma 1. *Given n aircraft with their approach entry times t_i and ideal landing times τ_i, the platoon list $\mathcal{P}$ can be computed in $O(n \log n)$ time.*

Proof. Denote the aircraft $\{1, \ldots, n\}$ in the order as they enter the approach. Construct the lists $T_1 = \{t_1, t_2, \ldots, t_n\}$ and $T_2 = \{\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_n}\}$, sorted from lowest to highest value. As there is no overtaking, the order of the aircraft in the approach is given by T_1. Now, we need to partition the aircraft into platoons.

Consider the maximum landing time τ_{i_n} and the corresponding entry time t_{i_n}. Aircraft i_n and all the aircraft following it (*i.e.*, aircraft $i_n+1, i_n+2, \ldots, n$), form a platoon with aircraft i_n in the head. Put the platoon at the beginning of list $\mathcal{P}$ and delete the entries corresponding to aircraft $i_n, i_n+1, \ldots, n$ from T_1 and T_2. Repeat until T_1 and T_2 are empty. By construction, each element in $\mathcal{P}$ is a platoon of aircraft sorted by their approach entry times, and the platoons themselves are sorted by the entry times of their head aircraft. The constructive part takes $O(n)$ time as each aircraft is deleted from T_1 and T_2 only once. The bottleneck is the sorting of lists T_1 and T_2, which leads to the total $O(n \log n)$ time for constructing the platoon list. □

The landing times of all aircraft in a platoon are equal to the landing time of the head aircraft in the platoon. Thus, after constructing the platoon list $\mathcal{P}$ we can find landing times r_i for all aircraft.

2.2. Approach Space Closure

Now, if the airspace was closed during the time interval $(T, T+L)$, all the aircraft scheduled to enter the approach at times that fall in this interval will be delayed until the airspace reopens at time $T+L$. Recall from Section 1.1 that the order in which the delayed aircraft enter the approach does not change. In Figure 1, the red interval represents the approach space closure. The aircraft affected by it shift their starting points to the end of the interval. Note that the interval can affect not only the aircraft with starting points falling in the closure interval, but also aircraft with starting points after the airspace is reopened again.

For a given length L of the airspace closure time interval, we want to find the best starting point T that minimizes the delay introduced into traffic. Note that there exists an optimal closure interval with the starting point T equal to one of the entry times t_i, or to $t_{\min}$ or $t_{\max}$; otherwise we can slide the interval to the left along the time axis without changing its length until we reach some t_i—this would not increase the delay, as no new aircraft would get affected, and the delay of already affected aircraft would only reduce.

Therefore, it suffices to only consider intervals starting at $t_1, \ldots, t_n$, plus the two special cases given by $t_{\min}$ and $t_{\max}$.

A naive solution is to consider all intervals $(t_i, t_i + L)$ separately, calculating the platoon list for each interval in $O(n)$ time and comparing the resulting values of $D(T, L)$; this leads to an overall $O(n^2)$-time algorithm.

The above solution is not efficient, as it involves recomputing the platoon list n times. For a more efficient algorithm we will introduce an upgraded version of the platoon list, storing each platoon in a tree (Section 2.4), and using colors to distinguish between how aircraft are affected by the closure. We will still consider intervals $(t_i, t_i + L)$, but in a sorted order, and incrementally update the platoon tree structure at every step.

2.3. Aircraft Flying through Rainbows

Before we describe the colors we use, let us give some definitions.

Call aircraft i *delayed* if it lands later than its scheduled landing time r_i, *i.e.*, if $r_i(T, L) > r_i$; otherwise, if $r_i(T, L) = r_i$, aircraft i is *undelayed*. Notice that we define an aircraft to be delayed or undelayed with respect to its scheduled landing time r_i, not with respect to its desired landing time τ_i.

Let $t_i(T, L) \in \{t_i, T + L\}$ denote the time when aircraft i enters the approach given that the airspace is closed during the interval $(T, T + L)$. Define an aircraft j to be *directly affected* by the closure interval $(T, T + L)$ if $t_j \in (T, T + L)$, *i.e.*, if $t_j(T, L) = T + L > t_j$ (note that j may still be undelayed, because it was following a slow aircraft even without the closure). Say that i is *indirectly affected* if it is delayed but is not directly affected by the closure: $r_i(T, L) > r_i$ and $t_i(T, L) = t_i \geq T + L$ (*i.e.*, landing of i is delayed because another aircraft in front of i is directly affected).

Now we can define seven aircraft colors with respect to the closure interval $(T, T + L)$ (Figure 3; see also Figure 1, right):

dark green aircraft i is an undelayed head of a platoon: $t_i(T, L) = t_i$ and $r_i(T, L) = r_i = \tau_i$,

red aircraft i is a delayed directly affected aircraft that would have been dark green (head of a platoon) if the airspace was not closed: $T < t_i < T + L$, $t_i(T, L) = T + L$, $r_i(T, L) > r_i = \tau_i$,

yellow aircraft i is a delayed aircraft that is indirectly affected by the closure, and that originally was dark green: $T + L < t_i(T, L) = t_i$, $r_i(T, L) > r_i = \tau_i$,

light green aircraft i is an undelayed aircraft that is not the head of a platoon: $t_i(T, L) = t_i$ and $r_i(T, L) = r_i > \tau_i$,

purple aircraft i is a delayed aircraft that is directly affected by the airspace closure, and that originally, before introducing the closure, was light green: $t_i \in (T, T + L)$, $t_i(T, L) = T + L$, $r_i(T, L) > r_i > \tau_i$,

orange aircraft i is a delayed aircraft that is indirectly affected by the closure, and that originally was light green: $t_i(T, L) = t_i > T + L$ and $r_i(T, L) > r_i > \tau_i$,

blue aircraft i is an undelayed aircraft that is directly affected: $T < t_i < T + L$, $t_i(T, L) = T + L$, $r_i(T, L) = r_i$; an aircraft can be blue only if originally, before the closure, it was light green.

	undelayed, unaffected	undelayed, directly affected	delayed, directly affected	delayed, indirectly affected
was platoon head		impossible		
was following aircraft				

Figure 3: Overview of the colors, based on the properties of the aircraft before the closure and their interaction with the current closure interval.

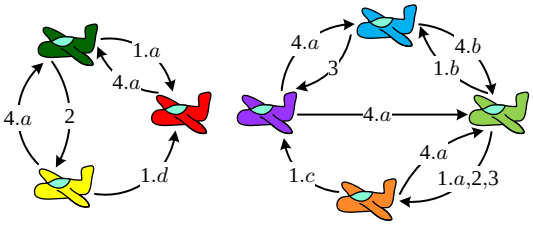


Figure 4: Change of the aircraft colors.

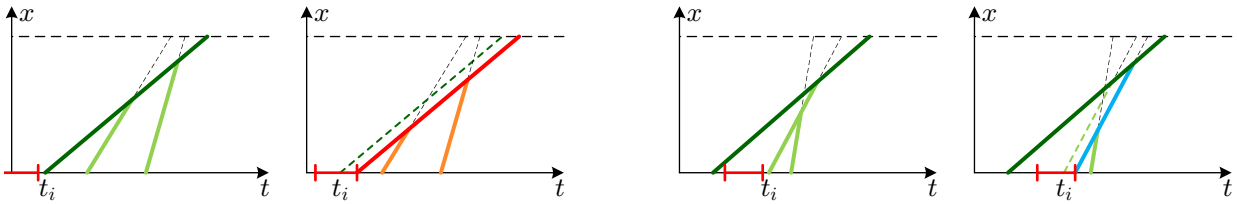
The colors indicate whether an aircraft is delayed or not, and if originally (before introducing the approach closure) the aircraft headed a platoon or not. A dark green aircraft (head of a platoon) can, possibly, become red or yellow, but never blue, purple, light green or orange; similarly, a light green aircraft (not a head of a platoon), cannot become red or yellow (refer to Figure 3).

Consider sliding the closure interval by an infinitesimally small amount from $(T-\varepsilon, T+L-\varepsilon)$ to $(T, T+L)$. There are four types of events that can happen during sliding, and that cause aircraft to change their colors (refer to Figure 4):

1. the right end of the interval passes the approach entry time of some aircraft i, *i.e.*, when $t_i(T,L)$ changes from t_i to $T+L$. If i was (a) dark green, then it becomes red, and the rest of the aircraft in the platoon following i become orange (Figure 5a), (b) light green, then it becomes blue (Figure 5b), (c) orange, then it becomes purple (Figure 5c), (d) yellow, then it becomes red (Figure 5d).

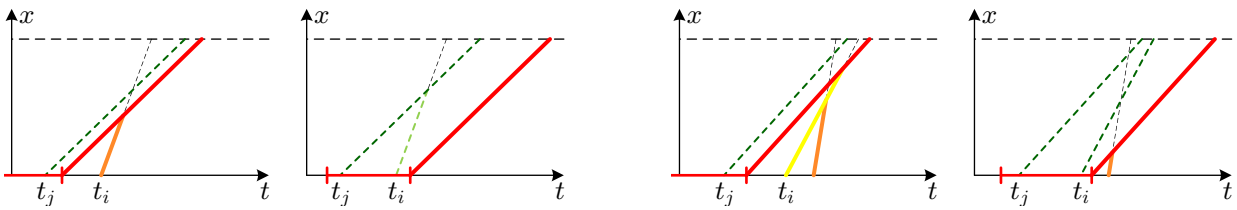
2. two consecutive platoons p_i, with red head aircraft i, and p_j, with dark green head aircraft j, merge, *i.e.*, $T+L+\tau_i-t_i > \tau_j$. This event changes the color of j to yellow and the rest of the aircraft from platoon p_j from light green to orange (Figure 6).

3. the interval pushes some aircraft out of a platoon. More precisely, when i is a blue aircraft in a platoon p_j with dark green head aircraft j, *i.e.*, $t_j < T < t_i < T+L$, and $T+L+\tau_i-t_i = \tau_j$, then new platoon p_i with head aircraft i is separated from platoon p_j. This event changes the color of i from blue to purple and the colors of the rest of the aircraft in the new platoon from light green to orange (Figure 7).



(a) dark green to red (following aircraft change from light green to orange)

(b) light green to blue



(c) orange to purple

(d) yellow to red

Figure 5: Aircraft change colors when the right end of the interval passes their approach entering times.

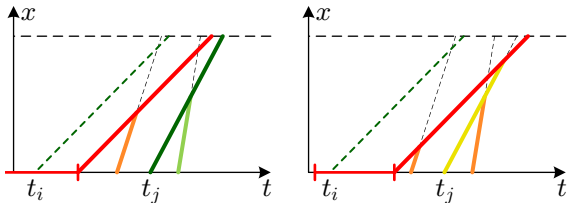


Figure 6: Two platoons merge.

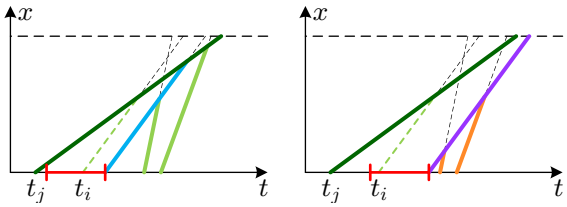
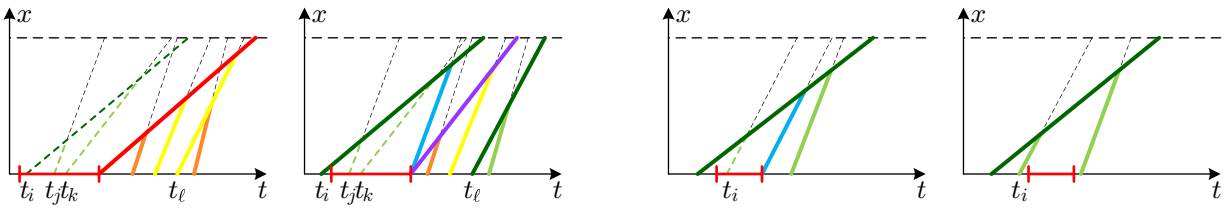


Figure 7: Subplatoon is pushed out from a platoon.



(a) (b)

Figure 8: Left end of the interval passes an aircraft approach entry time.

4. when the left end of the closure interval passes approach entry time of aircraft i, it changes its color back to light or dark green. Its color right before the change can be red, purple, or blue.

   (a) if i was red then it becomes dark green, if it was purple then it becomes light green. There can be the following changes in other aircrafts' colors (Figure 8a): (i) if there were yellow aircraft, some of them can become dark green, (ii) if there were orange aircraft, they can become light green, (iii) if there were other purple aircraft, some of them can become blue.

   (b) if i was blue, then it becomes light green (Figure 8b).

In our algorithm, we will slide the closure interval $(T, T+L)$ to the right, starting from $T = t_{\min} - L$ to $T = t_{\max}$, tracking all the events, updating the colors of the aircraft, and updating the delay function $D(T, L)$.

2.4. Platoon Tree Structure

We now introduce the *platoon tree structure* that will allow us to calculate $D(T, L)$ efficiently while sliding the closure interval. Instead of representing each platoon as a sorted list as before, we now store it in a tree with nodes sorted by increasing approach entry times top-to-bottom and left-to-right among siblings, and also sorted by desired speeds increasing top-to-bottom but decreasing left-to-right among siblings. Specifically:

- the root of the tree is the head of the platoon,
- if node j is a child of node i then $t_i < t_j$ and $v_i < v_j$
- if j is the left sibling of i then $t_i > t_j$ but $v_i < v_j$. Moreover, the same inequalities hold for i and any descendant j' of j: $t_i > t_{j'}$ but $v_i < v'_j$.

For example, Figure 9 shows a platoon tree that corresponds to a platoon consisting of aircraft with speeds $\{40, 65, 70, 90, 85, 80, 67, 60, 70\}$. We remark that platoon trees are in no way related to the green trajectories trees in the tx-space (see, *e.g.*, Figure 1).

The platoon tree can be constructed from a platoon list in linear time by scanning the list of aircraft in increasing order of t_i (*i.e.*, starting from the head) and making node $i+1$ the child of i if $v_{i+1} > v_i$. Otherwise, *i.e.*, if $v_{i+1} < v_i$, we go up the tree to the first node j on the i-to-root path that has $v_j < v_{i+1}$ and make $i+1$ the rightmost child of j. Such a node will always exist as the root is slower than any aircraft in the platoon.

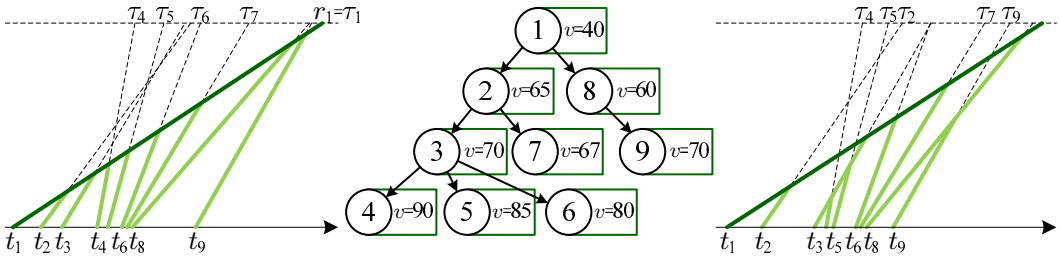


Figure 9: Platoon tree for aircraft with desired velocities $(40, 65, 70, 90, 85, 80, 67, 60, 70)$, listed in the order of increasing entry times. Note that the same platoon tree may correspond to different trajectory trees in the tx-space (shown left and right).

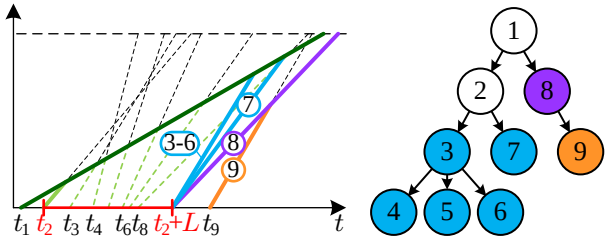


Figure 10: The property of the platoon tree structure illustrated on the platoon from the left example in Figure 9. Left: Trajectories in tx-space after the closure $(t_2, t_2 + L)$. Right: Colored nodes of the tree are affected by the interval. They form three subtrees, and there are three subplatoons entering the approach at the same time $t_2 + L$.

Any node is visited only once, as the construction is equivalent to traversing the tree in the depth-first-search order. Therefore, the total time it takes to construct a platoon tree from a platoon list is $O(n)$.

The platoon tree structure has one nice property that we are going to use later on: consider tree structure $\mathcal{T}$ and the corresponding platoon p with aircraft $\{1, 2, \ldots, j\}$. Let the closure interval $(T, T + L)$ for some $T = t_i$ contain the entry times $\{t_{i+1}, \ldots, t_{i+k}\}$ of k aircraft from p ($t_1 < t_{i+1}$ and $t_{i+k} \leq t_j$). Mark the nodes in $\mathcal{T}$ that correspond to the aircraft directly affected by the interval (see Figure 10). Some of these marked nodes occur to be roots of subtrees that correspond to the subplatoons starting at point $T + L$. The number of these subtrees is equal to the number of subplatoons starting at $T + L$, and the value of the delay function $D(T, L)$ depends only on the sizes of these subtrees and the landing times of their root nodes. This property allows us to update the platoons fast after the left end of the closure interval jumps over the approach entry time of the head h of a platoon (thus breaking the platoon)—the new platoons are simply the subtrees rooted at children of h.

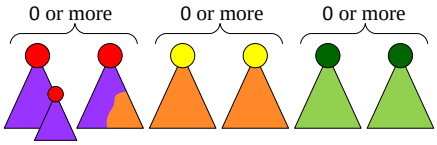
2.5. Rainbow Algorithm

In this section we present an $O(n \log n)$-time algorithm to find an interval of given length L that minimizes the total delay $D(T, L)$.

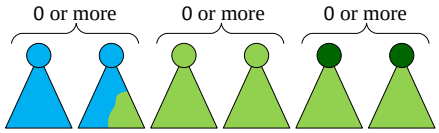
Let $\mathcal{F}_0$ be the forest consisting of original platoon trees before introducing the approach closure. Let m be the total number of platoons formed by n aircraft. Then $\mathcal{F}_0$ contains m trees and n nodes in total. The trees in $\mathcal{F}_0$ are sorted by the approach entry times of their head aircraft.

We will slide the closure interval from $(t_{\min}, t_{\min} + L)$ to $(t_{max}, t_{\max} + L)$, update the platoon forest structure, and calculate the delay function at every step. Let forest $\mathcal{F}(T)$ describe the state of the system for a closure interval $(T, T + L)$. It contains all the aircraft with approach entry times greater than T. All the aircraft that enter the approach before T cannot be delayed, therefore we do not need to keep track of all of them in $\mathcal{F}(T)$. The only aircraft among them, that we need to keep track of, is the last head of a platoon that enters the approach before T, as it still can affect some of the aircraft in $\mathcal{F}(T)$.

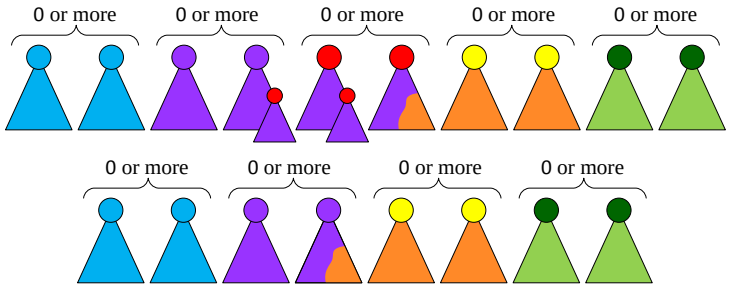
A tree in $\mathcal{F}(T)$ can be the same as some original tree in $\mathcal{F}_0$, be a subtree of some tree in $\mathcal{F}_0$, or consist of some trees (and subtrees) from $\mathcal{F}_0$ merged together. For a closure interval $(T, T + L)$ forest $\mathcal{F}(T)$ can be



(a) Closure interval is in between two platoons. Forest structure $\mathcal{F}(T)$ consists of zero or more trees with red, yellow, and dark green roots. If there are no trees with red roots, then $\mathcal{F}(T)$ can only consist of trees with dark green roots. In this case there is no delay.



(b) Closure interval falls in the middle of a platoon, but there are no delayed aircraft. Forest structure $\mathcal{F}(T)$ consists of zero or more trees with blue, light green, and dark green roots.



(c) Closure interval falls in the middle of a platoon. Forest structure $\mathcal{F}(T)$ consists of zero or more trees with blue, purple, red, yellow, and dark green roots. The last tree with a red root, or the last tree with a purple root (if there are no trees with red roots), can contain orange nodes. The delay is determined by the trees with purple, red, and yellow roots.

Figure 11: Three states of forest structure $\mathcal{F}(T)$.

obtained from $\mathcal{F}_0$ with the following actions:

1. delete all the nodes with approach entry times less or equal to T from $\mathcal{F}_0$;
2. determine the colors of the roots of the trees left in $\mathcal{F}_0$;
3. merge the trees with red roots with the previous trees with red or purple roots if needed.

We merge some trees with red or purple roots to, later, speed up the calculation of $D(T, L)$. As mentioned earlier, to calculate the delay function we need to know the sizes of the trees and the landing times of their roots. If a red root of some tree $\mathcal{T}_i$ has landing time $r_{\mathcal{T}_i}(T, L)$ that is equal to the landing time $r_{\mathcal{T}_j}(T, L)$ of an earlier tree $\mathcal{T}_j$ (with a slower heading aircraft), merging these trees, by making $\mathcal{T}_i$ a subtree of $\mathcal{T}_j$, will allow us to calculate the contribution to $D(T, L)$ of both trees in one step.

To merge tree $\mathcal{T}_i$ with a previous tree $\mathcal{T}_j$, we descend from the root of $\mathcal{T}_j$ along the rightmost chain, comparing the velocities of the nodes in the chain with the velocity of the root of $\mathcal{T}_i$. Similarly to building the platoon tree structure (recall from Section 2.4), we insert the root of $\mathcal{T}_i$ as a right-most child of the last node whose velocity is less than the velocity of $\mathcal{T}_i$'s root. This operation can be implemented with a binary search along the rightmost root-to-leaf path, therefore it will take $O(\log n)$ time per each merge.

We will only apply the merge operation when $\mathcal{T}_i$ has a red root, and $\mathcal{T}_j$ has a red or a purple root. Trees with purple roots already have the property that they are ordered in the decreasing order of their roots' velocities, by construction of the platoon tree structure. And, unlike the red roots, yellow roots may change their color back to dark green while still be entering the approach after the closure (Section 2.3). Thus, if we merge trees with yellow roots, we may need to split them later. Red nodes, however, can only become dark green when they get deleted from $\mathcal{F}(T)$, therefore, a tree with a red root can be merged with an earlier tree only once.

```
1: function RAINBOW
2:     initialize F                          ▷ at the beginning F(T) contains all the aircraft
3:     T_min ← t_min − L, D_min ← ∞          ▷ start with the closure interval (t_min − L, t_min)
4:     for T in {t_min, t_1, ..., t_n} do    ▷ move the closure interval to the next t_i
5:         if T > t_max then return T_min, D_min
6:         update F(T)
7:         D ← find D(T, L)
8:         if D < D_min then
9:             T_min ← T, D_min ← D
10:        end if
11:    end for
12:    return T_min, D_min
13: end function
```

Figure 12: Rainbow algorithm iterates over all intervals $(t_i, t_i + L)$ for all $t_{\min} \le t_i \le t_{\max}$ and returns the one with the minimum value of the delay function.

All the nodes in every tree of $\mathcal{F}(T)$ have the same landing times as their roots. Therefore, to calculate the value of delay function $D(T, L)$ we need to know the number of nodes in each tree, and the new delayed arrival time of their roots. In every node i of $\mathcal{F}(T)$ we store t_i and τ_i.

Figure 11 shows three possible colorings of the trees in $\mathcal{F}(T)$. If the closure interval $(T, T + L)$ falls in between two platoons, (*i.e.*, T is greater or equal to the approach entry time of the last aircraft in one platoon, and it is less than the entry time of the head of the next platoon), then $\mathcal{F}(T)$ consists of zero or more trees with red roots, possibly followed by several trees with yellow roots, and several trees with dark green roots in the end (Figure 11a). The red roots correspond to the heads of the platoons that are directly affected by the closure, their starting time is $T + L$. All the following aircraft in these platoons are purple or orange, they have the same landing times as their roots. If head aircraft i of some platoon is red, and its landing time is

$$r_i(T, L) = T + L + \tau_j - t_j > T + L + \tau_i - t_i \,,$$

i.e., this platoon follows and lands at the same time as an earlier platoon with a slower head aircraft j, then the tree with root i is, actually, a subtree of the tree with root j due to the merge rule.

If the closure interval falls in the middle of a platoon (*i.e.*, T is greater or equal to the approach entry time of the head aircraft of a platoon, and it is less than the entry time of the last aircraft in it), then the first several trees in $\mathcal{F}(T)$ are subtrees of the platoon tree corresponding to this platoon. There are two cases for the state of $\mathcal{F}(T)$. First, if the closure interval does not push any subplatoons out of the current platoon, *i.e.*, there are aircraft with entry times shifted to $T + L$, but their landing times are not affected by the closure, then there is no delay, and $\mathcal{F}(T)$ consists of zero or more trees with blue roots, followed by zero or more trees with light green roots, and the rest are the trees with dark green roots that correspond to the undelayed platoons following the current platoon (Figure 11b). In this case there is no delay, we can stop the algorithm and report a zero-delay position for the closure. Therefore, in what follows we will assume that this case does not occur[4]. Second, if the closure interval does push some subplatoons out of the current platoon, then $\mathcal{F}(T)$ consists of zero or more trees with blue roots, followed by several trees with purple roots, then with red roots, yellow roots, and dark green roots (Figure 11c). Trees with purple, red, and yellow roots contain the aircraft that are delayed. Some of the trees with red roots are merged to account for the same landing times of the corresponding platoons. Notice that only the last tree with a purple root and the trees with red roots can contain other trees with red roots as subtrees.

Details of the Algorithm. The outline of the algorithm is presented in Figure 12. Without loss of generality we assume that $t_{\min} < t_1$. One iteration of the **for**-loop corresponds to sliding the closure interval from

[4]Our algorithm can be modified to report all the closure intervals that correspond to zero-delay. But here we concentrate on a more interesting case when there is no zero-delay closure.

$(t_{i-1}, t_{i-1}+L)$ to (t_i, t_i+L), during which a number of color-changing events (described in Section 2.3) can occur. The algorithm updates the forest data-structure and recomputes the value of the delay function, processing all such events at the same time in the following way.

During the **update** $\mathcal{F}(T)$ subroutine in line 6, the leftmost root i is removed from $\mathcal{F}(T)$ and its children move one level up to become roots of their trees. Next, the colors of some nodes get updated. Specifically, node i, before the removal from $\mathcal{F}(T)$, can be blue, purple, red or dark green (it cannot be yellow or orange, as all aircraft of these colors must follow some delayed aircraft, which has to be in $\mathcal{F}(T)$ before such nodes), so after sliding the closure interval from $(t_{i-1}, t_{i-1}+L)$ to (t_i, t_i+L) and removing i from $\mathcal{F}(T)$, the following happens (cf. Figure 4):

- some yellow and dark green roots can become red (if the right end of the closure interval slides over the approach entry times of these aircraft). However red roots cannot become yellow, they only change their color back to dark green once they are removed from $\mathcal{F}(T)$;
- some dark green roots can become yellow, or some yellow roots can become dark green;
- some blue roots in $\mathcal{F}(T)$ can become purple (if the right end of the closure interval "pushes" some blue subplatoons out of the current platoon), or some purple roots can become blue (if there were no blue trees and several purple trees in $\mathcal{F}(T)$, and the removal of i caused some purple subplatoons to fall back under the current platoon).

The **update** subroutine does not update $\mathcal{F}(T)$ for every color-changing event one by one, but does it once, taking into account all the events that happened during the shift of the closure interval. Moreover, to make this update efficient we do not explicitly store the colors in the nodes of $\mathcal{F}(T)$, but use few variables that point to the first or the last root of a specific color in $\mathcal{F}(T)$. For clarity, in the following, we will assume the general case of the structure of $\mathcal{F}(T)$ shown in Figure 11c, where all the colors are present (it will be easy to account for special cases where some of the colors are absent). Specifically, the variables that we are going to be using will be indices of aircraft or indices of platoons:

1. Let p_c be the index of the original platoon from $\mathcal{F}_0$ that contains aircraft i, and let i_{p_c} be the index of the head aircraft of p_c; $p_c \in \{1, \ldots, m\}$, where m is the total number of platoons in $\mathcal{F}_0$, and $i_{p_c} \in \{1, \ldots, n\}$, where n is the total number of aircraft.

2. Let $\mathcal{T}_r$ be a pointer to the last tree in $\mathcal{F}(T)$ whose root is red. If there are no trees with red roots in $\mathcal{F}(T)$, then $\mathcal{T}_r$ will point to the last tree with a purple root. Recall that we assume that there is no zero-delay closure, and therefore there will always be a tree in $\mathcal{F}(T)$ with a red or purple root.

3. Let $\mathcal{T}_y$ be a pointer to the last tree in $\mathcal{F}(T)$ with a yellow root. If there are no trees with yellow roots, then $\mathcal{T}_y$ will point to the last delayed tree in $\mathcal{F}(T)$ (its root will be red, or purple, if there are no trees with red roots).

4. Let i_p, i_y, and i_g be the indices of the first purple, yellow, and dark green root in $\mathcal{F}(T)$ respectively; $i_p, i_y, i_g \in \{1, \ldots, n\}$, where n is the total number of aircraft. If there are no purple roots in $\mathcal{F}(T)$, then i_p will be the index of the first red root. If there are no trees with yellow roots in $\mathcal{F}(T)$, then i_y will be equal to i_g. And if there are no dark green roots, then $i_g = n+1$.

Knowing the ranges of the colors in $\mathcal{F}(T)$, and knowing the sizes of the corresponding trees, is enough to calculate the total delay. The **update** procedure is then, first, delete the leftmost node in $\mathcal{F}(T)$, and, second, update the values of p_c, $\mathcal{T}_r$, $\mathcal{T}_y$, i_p, i_y, and i_g (see Figure 13):

find $\mathcal{T}_r$: we can decide which yellow or dark green roots become red by comparing their starting times t_j with the right end of the closure interval $T+L$. Because the trees in $\mathcal{F}(T)$ are ordered by the starting times of their roots, we can find the last red root in $\mathcal{F}(T)$ with a binary search in $O(\log n)$ time (see the concluding remark at the end of this section for a brief discussion on the implementation of the binary search).

merge $\mathcal{T}$ with $\mathcal{T}$.prev: we can find the position along the rightmost chain in $\mathcal{T}$.prev where to insert the root of $\mathcal{T}$ with a binary search (see the remark at the end of this section for a brief discussion on the

```
1: procedure update F(T)
2:     delete the leftmost root from F(T)              ▷ its children become new roots
3:     T_r^old ← T_r                  ▷ save the old pointer to the last tree with a red root
4:     find T_r                                ▷ find trees in F(T) whose roots become red
5:     for T ← T_r^old.next to T_r do
6:         if T.root is slower than T.prev.root then
7:             merge T with T.prev
8:         end if
9:     end for
10:    find T_y                          ▷ find trees in F(T) whose roots become yellow
11:    find i_p                             ▷ find trees in F(T) whose roots are purple
12:    i_y ← T_r.next.root.index                ▷ find the index of the first yellow node
13:    i_g ← T_y.next.root.index           ▷ find the index of the first dark green node
14: end procedure
```

Figure 13: Subroutine **update** $\mathcal{F}(T)$.

implementation of the binary search) as the nodes in the chain are ordered by their velocities. Every tree with a red root can be merged with a previous tree at most once, therefore, in total, all the merging operations will take $O(n \log n)$ time.

find $\mathcal{T}_y$: we can identify which yellow roots become dark green or which dark green roots become yellow by comparing their landing times r_j with the landing time $r_{\mathcal{T}_r}(T, L) = \tau_{\mathcal{T}_r}(T, L)$ of the root of tree $\mathcal{T}_r$ (the last tree with a red root in $\mathcal{F}(T)$). We can find the last yellow root in $\mathcal{F}(T)$ with a binary search over all trees that come after $\mathcal{T}_r$.

find i_p: if there are blue or purple trees in $\mathcal{F}(T)$, then the closure interval falls in the middle of the current platoon p_c. Denote the arriving time of its head aircraft by r_{p_c}, which is equal to its ideal landing time τ_{p_c}. This aircraft is not delayed anymore and was removed from $\mathcal{F}(T)$ in this or one of the previous iterations. We can identify which trees in $\mathcal{F}(T)$ have blue roots and which trees have purple roots by comparing their arrival time $r_j(T, L) = T + L + \tau_j - t_j$ with $r_{p_c} = \tau_{p_c}$ for every root j of the trees that correspond to the subplatoons of p_c that are still in $\mathcal{F}(T)$. The roots are purple if $r_j(T, L) > r_{p_c}$, and blue otherwise. Again, we can find i_p with a binary search in $O(\log n)$ time.

Therefore, in total, the **update** $\mathcal{F}(T)$ subroutines in line 5 of the algorithm will take $O(n \log n)$ time.

Finally, function **find** $D(T, L)$ in line 7 of the algorithm computes the delay function for closure interval $(T, T + L)$. Its outline is shown in Figure 14. To be able to compute the delay in constant time we introduce two arrays A and C:

1. Array $C[1..m]$ has a cell for each platoon in $\mathcal{F}_0$. $C[j]$, corresponding to platoon p_j, contains value

$$C[j] = \sum_{u=1}^{j} k_u r_u \,,$$

   where k_u is the size of platoon p_u, and r_u is the landing time of the head aircraft of p_u.

2. Array $A[1..n]$ has a cell for each aircraft. To populate the cells of A we will need to temporarily merge some of the trees in $\mathcal{F}_0$, using the same rules as when merging the trees with red roots, but now applying these rules to all the trees. Consider two consecutive platoons p_i and p_{i+1} and the corresponding trees $\mathcal{T}_i$ and $\mathcal{T}_{i+1} \in \mathcal{F}_0$. Let v_i and v_{i+1} be the velocities of the head aircraft of p_i and p_{i+1} correspondingly. If $v_i < v_{i+1}$, then find the first node along the right-most chain in $\mathcal{T}_i$ whose velocity is not less than v_{i+1}, and make the root of $\mathcal{T}_{i+1}$ its right sibling. Repeat this process until all trees in $\mathcal{F}_0$ are ordered in the decreasing order of the velocities of their roots. Denote the resulting forest as $\mathcal{F}'$.

   We can also construct $\mathcal{F}'$ by applying the rules used to build a platoon tree (recall Section 2.4) to all n aircraft, instead of just to aircraft of one platoon.

1: **function find** $D(T,L)$
2: **compute** $D_d(T,L)$ ▷ contribution to $D(T,L)$ by delayed aircraft
3: **compute** $D_0(T,L)$ ▷ contribution to $D(T,L)$ by undelayed aircraft
4: **return** $D_d(T,L)+D_0(T,L)$
5: **end function**

Figure 14: Function **find** $D(T,L)$.

Now consider any node $j \in \mathcal{F}'$, consider trees $\mathcal{T}_1^j, \mathcal{T}_2^j, \ldots, \mathcal{T}_\ell^j$ that are left in $\mathcal{F}'$ after removing the nodes $1, 2, \ldots, j-1$ from it. Set

$$A[j] = \sum_{u=1}^{\ell} k_u(\tau_u - t_u)\,,$$

where k_u is the size of $\mathcal{T}_u^j$, t_u and τ_u are the approach entry time and the desired landing time of the aircraft corresponding to the root of $\mathcal{T}_u^j$.

The construction of arrays A and C can be done in initialization step in linear time.

compute $D_d(T,L)$: the contribution to the delay function of the delayed aircraft is

$$D_d(T,L) = \sum_{u\in\substack{\text{purple,}\\ \text{red}}} k_u(T+L+\tau_u-t_u) + \sum_{u\in\text{yellow}} k_u(T+L+\tau_{\mathcal{T}_r}-t_{\mathcal{T}_r}) =$$

$$\left(\sum_{u\in\substack{\text{purple,}\\ \text{red,}\\ \text{yellow}}} k_u\right)(T+L) + \sum_{u\in\substack{\text{purple,}\\ \text{red}}} k_u(\tau_u-t_u) + \sum_{u\in\text{yellow}} k_u(\tau_{\mathcal{T}_r}-t_{\mathcal{T}_r})\,,$$

where the summation is over all trees $\mathcal{T}_u$ in $\mathcal{F}(T)$ with purple, red, and yellow roots, k_u is the size of the corresponding tree, t_u and τ_u are the approach entry time and the landing time of this root, and $t_{\mathcal{T}_r}$ and $\tau_{\mathcal{T}_r}$ are the approach entry time and the landing time of the last tree with a red root. To calculate the first term of $D_d(T,L)$ we need to know the sum of the sizes of all the delayed trees. In procedure **update** $\mathcal{F}(T)$ we calculated i_p and i_g—the indices of the first purple and dark green roots. Then the total number of nodes in the trees with purple, red, and yellow roots is $(i_g - i_p)$, and thus,

$$\left(\sum_{u\in\substack{\text{purple,}\\ \text{red,}\\ \text{yellow}}} k_u\right)(T+L) = (i_g-i_p)(T+L)\,.$$

The second term of $D_d(T,L)$ can be found using array A,

$$\sum_{u\in\substack{\text{purple,}\\ \text{red}}} k_u(\tau_u-t_u) = A[i_p]-A[i_y]\,,$$

as all the nodes in purple and red trees have indices between i_p and i_y-1. And the third term of $D_d(T,L)$ is

$$\sum_{u\in\text{yellow}} k_u(\tau_{\mathcal{T}_r}-t_{\mathcal{T}_r}) = (i_g-i_y)(\tau_{\mathcal{T}_r}-t_{\mathcal{T}_r}).$$

as all the nodes in yellow trees have indices between i_y and i_g-1. Thus, the contribution of the trees with purple, red, and yellow roots to the delay can be calculated in constant time by formula

$$D_d(T,L) = (i_g-i_p)(T+L) + A[i_p] - A[i_y] - (i_g-i_y)(\tau_{\mathcal{T}_r}-t_{\mathcal{T}_r})\,.$$

compute $D_0(T, L)$: the rest of the aircraft that have not been accounted for so far are undelayed. The contribution to the delay function of all the platoons that have already been completely removed from $\mathcal{F}(T)$ is equal to $C[p_c - 1]$. The contribution of all the platoons with dark green roots that are still present in $\mathcal{F}(T)$ is $C[m] - C[\mathcal{T}_y.\text{index}]$, where $\mathcal{T}_y.\text{index}$ is the index of the platoon corresponding to the last tree with a yellow root. Therefore, the contribution of the undelayed aircraft from the current platoon is $(i_p - i_{p_c})r_{p_c}$, where i_p is the first purple aircraft, i_{p_c} is the head of the current platoon, and $r_{p_c} = \tau_{p_c}$ is its landing time. Therefore,

$$D_0(T, L) = C[p_c - 1] + C[m] - C[\mathcal{T}_y.\text{index}] + (i_p - i_{p_c})\tau_{p_c} ,$$

Then the value of the delay function for the interval $(T, T + L)$ is

$$D(T, L) = D_d(T, L) + D_0(T, L) ,$$

and it can be calculated in constant time at every step of the algorithm. There are $n+1$ iterations of the algorithm, that in total take $O(n \log n)$ time to execute.

Theorem 2. *Rainbow algorithm finds an interval of given length that minimizes the delay in $O(n \log n)$ time.*

For the query version of the problem, studied in next section, we need to build $D(T, L)$ for fixed T but varying L; this can also be accomplished in $O(n \log n)$ time, similarly to the above. In this version of the problem if an aircraft is delayed it will never become undelayed during the execution of the algorithm, *i.e.*, the events that cause aircraft to change their colors only include the first three types described in Section 2.3. To update the forest structure $\mathcal{F}(T)$ we no longer delete the leftmost root. The rest of the algorithm is similar to the algorithm presented in this section.

Remark on Implementation. In the **update** $\mathcal{F}(T)$ procedure of our algorithm we perform three binary searches on a list of roots in platoon forest structure $\mathcal{F}(T)$ when we search for $\mathcal{T}_r$, $\mathcal{T}_y$, and i_p. Such binary searches can be implemented using three balanced binary search trees (for example, AVL trees), containing values t_j of roots of trees in $\mathcal{F}(T)$ for searching for $\mathcal{T}_r$, values r_j of roots for searching for $\mathcal{T}_y$, and values $\tau_j - t_j$ for searching for i_p. These trees can be initialized and maintained during the execution of the algorithm without affecting the asymptotic running time.

We also use a binary search on the rightmost root-to-leaf chain in a tree with a red (or purple) root when merging two platoon trees. To implement such a search we can again use balanced binary search trees built on velocities of the aircraft in the rightmost chains of trees in $\mathcal{F}(T)$. These trees will need to be updated during the merge operation. During merging of two platoon trees, one binary tree, that corresponds to the left platoon tree, will be split, and one of the resulting parts will be concatenated with another binary search tree, that corresponds to the right platoon tree. After the split-concatenate operation the trees can be rebalanced in $O(\log n)$ running time. Therefore, binary searches in the rightmost root-to-leaf chains can be performed in $O(\log n)$ running time per each search.

3. Ego Query

We now consider the query version of delay minimization, *i.e.*, answering queries of the type "Given the length L of the ego, report the best closure start moment T". We build an $O(n^2\alpha(n))$-complexity function, where $\alpha(n)$ is the inverse Ackermann's function, to answer such queries in $O(\log n)$ time using $O(n^2 \log n)$ preprocessing time.

Extending the idea from the previous section, we start from building the *delay diagram* (Figure 15), the subdivision of the LT-space into cells such that the delay $D(T, L)$ is given by the same function within a cell. We build the diagram separately in each strip between the lines $T = t_{i-1}$ and $T = t_i$; clearly, the function $D(T, L)$ stays combinatorially the same for all $T \in (t_{i-1}, t_i)$ (however not all lines $T = t_i$ are necessarily edges of the diagram, so we may actually compute a slightly finer subdivision than needed). We fix an arbitrary point $T \in (t_{i-1}, t_i)$ and increase L watching for *events* when $D(T, L)$ changes; the events and updates are

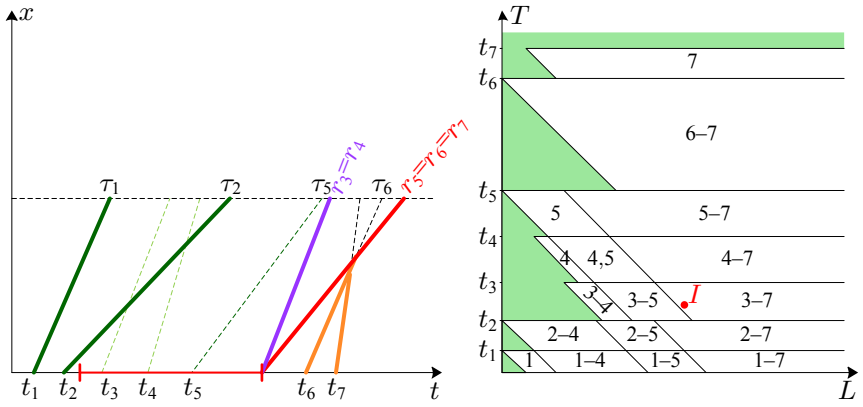


Figure 15: The diagram (right) for the instance on the left. (Green is the 0-impact region; see Section 4.) The aircraft influenced by the closure are indicated in each cell. Point I corresponds to the closure interval from the left.

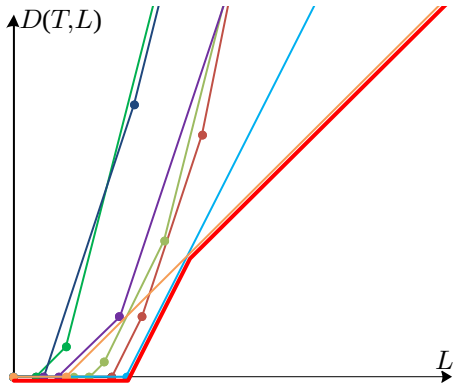


Figure 16: The lower envelope (red) for the instance from Figure 15. Every color corresponds to a horizontal line $T = t_i$ in the diagram (the functions for diagonal edges of the diagram are not shown because the minimum is attained on the horizontal edges).

analogous to those considered in the previous section (the only difference is that in the previous section we kept the length L fixed and varied the starting time T, while now we have fixed T and vary L). We spend $O(n \log n)$ time per strip, constructing the diagram in overall $O(n^2 \log n)$ time (note that the diagram may have quadratic complexity overall, *e.g.*, if the trajectories of aircraft pairwise do not cross, *i.e.*, if each aircraft is a platoon by itself).

Since $D(T, L)$ changes linearly within each cell of the diagram, for any L, the optimal T will lie on an edge of the diagram. We graph $D(T, L)$ as functions of L for all diagram edges on a single plot (Figure 16); since the diagram has $O(n^2)$ complexity, there are $O(n^2)$ segments on the plot, and the optimal T as a function of L is the lower envelope of the segments. The function has $O(n^2\alpha(n))$ complexity and can be constructed from the segments in $O(n^2 \log n)$ time [15]. Then the query can be answered in $O(\log n)$ time by locating L and reporting the value of the function.

Theorem 3. *The lower envelope function of $O(n^2\alpha(n))$ complexity can be calculated in $O(n^2 \log n)$ time that will allow to find the optimal closure time T for a given closure length L in $O(\log n)$ time.*

4. Algorithms for Harmless VIPs

In the LT-space, the cells with 0-level of the function $D(T, L)$ correspond to closure intervals $(T, T+L)$ that do not introduce any delay into the traffic. Obviously, the T-axis belongs to the 0-level ($D(T, L) = 0$ for

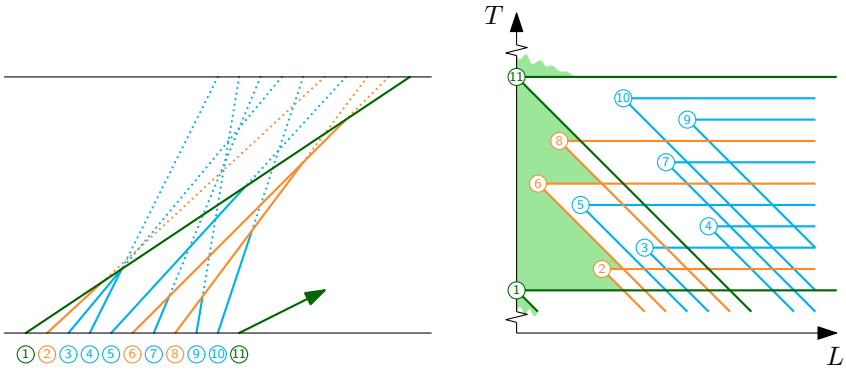


Figure 17: Left: A platoon with 10 aircraft; $i=1, h=11$. Right: the corresponding sub-Christmas-tree. Platoon heads are green, and other aircraft that influence the shape of the tree are orange; the other aircraft are blue. Circles are the points $(\tau_i - \tau_j, t_j)$.

$L=0$) and for every T there is a maximum L for which $D(T,L)=0$. Thus, the right boundary of the level is a T-monotone curve, and the level is (the right half of) a tree with the T-axis as the trunk. Moreover, since the 0-level is a union of cells of the delay diagram and edges of the diagram are straight-line segments, the right boundary of the level is a polygonal curve. We therefore refer to the 0-level as the *Christmas tree* (Figure 17; see also Figure 15).

Clearly, the Christmas tree can be constructed in $O(n^2)$ time using results from the previous section. In this section we show how to compute the tree in linear time (after platoons in the absence of the closure have been identified, which can be done in $O(n \log n)$ time). The crucial observation is that the Christmas tree can be built platoon-by-platoon: if the closure interval $(T, T+L)$ contains the entry time t_i for a platoon head i, then i is delayed and (L,T) is not in the Christmas tree; thus, it is enough to consider only intervals lying between entry times of two consecutive heads. Moreover, the right boundary of the Christmas tree must touch the T-axis at t_i (since for T infinitesimally smaller than t_i we have $D(T,L)>0$ for infinitesimally small L). Hence, entry times of platoon heads split the Christmas tree into "sub-Christmas-trees" touching the T-axis at their bottoms and tops (Figure 17, right). In what follows we will focus on building the (sub-)tree for one platoon, headed by i, and assume that the closure starts between t_i and t_h where h is the head of the platoon that follows i's platoon (so the aircraft in i's platoon are $i, \dots, h-1$). That is, our goal will be to build the (sub-)tree in the wedge $T > t_i, T+L < t_h$.

For a given starting time $t_i < T < t_h$ of the closure, increase its length L until the first aircraft j gets delayed. If j is a head of a platoon, *i.e.*, j is aircraft h, then $T+L > t_h$. Else, if $j \in \{i+1, \dots, h-1\}$, then j is the first purple root. It cannot be yellow or red, as j is not a head of a platoon. It also cannot be orange, otherwise it would follow another delayed aircraft. Then, the following inequalities hold

$$T < t_j < T+L\,,$$

$$T+L+\tau_j - t_j > r_i = \tau_i\,.$$

That is, aircraft j is delayed if (L,T) lies in the wedge between the rays $T=t_j$ and $T+L=t_j+\tau_i-\tau_j$ emanating from the point $(\tau_i-\tau_j, t_j)$ (refer to Figure 17, right). Our (sub-)tree is the complement of the union of the wedges for all aircraft $i, \dots, h$. To build the tree observe that it is the Pareto envelope (set of undominated points) of the apexes of the wedges in a sheared copy of the LT-space. Since aircraft are sorted along the T axis, the envelope can be build in linear time: scan the apexes from top to bottom and for each apex test whether it lies to the left or to the right of the diagonal line through the previous apex; in the former case include it in the envelope, in the latter case throw it away.

Overall we obtain that the Christmas tree can be built in $O(n \log n)$ time. Using the tree, one can answer a query "Given T, report the largest L for which $D(T,L)=0$" in logarithmic time: locate T between t_i and t_{I+1} and shoot horizontal ray until leaving the Christmas tree.

Theorem 4. *The Christmas tree of $O(n)$ complexity can be built in $O(n \log n)$ time that will allow to find the maximum closure length L for a given closure time T in $O(\log n)$ time.*

5. Conclusion

VIPs live from favorable comparisons to other VIPs in the public eye. Unfortunately even in the current era of world-wide information availability, people tend to rely on subjective feelings (nurtured, *e.g.*, during election campaigns) when forming their opinions about VIPs. This is partially due to the absence of numerical criteria to access, with certainty, how good or bad a VIP is. In this paper we made an important step towards objective judgement of politicians by offering the first succinct measure of politicians egos—the requested length of airspace closure.

We presented efficient algorithms to minimize the impact of the airspace closure on commercial traffic. Moreover, we showed how to compute the impact as a function of the closure time and duration, from which the optimal closure can be inferred. In fact, the function also allows one to answer other questions about the closure, *e.g.*, what closure would have the *largest* impact on society (we deliberately did not elaborate on this, fearing that Herostratic VIPs may use our algorithms to show off how influential they can be).

Future work includes begging VIPs for funds to support research aiming at minimizing their harm.

Acknowledgements

We thank the anonymous reviewers for valuable advice and suggestions that led to significant improvements in the paper.

References

[1] I. Kostitsyna, M. Löffler, V. Polishchuk, Optimizing airspace closure with respect to politicians egos, in: Fun with Algorithms, Springer International Publishing, 2014, pp. 264–276.
[2] ATCOs, Personal communication (2013).
[3] SESAR, European ATM Master Plan (2012).
[4] T. Allard, Personal communication (2012).
[5] M. d. Berg, O. Cheong, M. v. Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications, 3rd Edition, Springer-Verlag, Santa Clara, CA, USA, 2008.
[6] A. May, Traffic Flow Fundamentals, Prentice Hall, 1990.
[7] W. McShane, R. Roess, E. Prassas, Traffic Engineering, Prentice Hall, 1998.
[8] F. L. Hall, Traffic stream characteristics, Traffic Flow Theory. US Federal Highway Administration.
[9] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, J. Luo, Detecting commuting patterns by clustering subtrajectories, Int. J. Comput. Geometry Appl. 21 (3) (2011) 253–282.
[10] B. S. Kerner, H. Rehborn, Experimental features and characteristics of traffic jams, Phys. Rev. E 53 (1996) R1297–R1300. doi:10.1103/PhysRevE.53.R1297.
[11] B. S. Kerner, H. Rehborn, Experimental properties of complexity in traffic flow, Phys. Rev. E 53 (1996) R4275–R4278. doi:10.1103/PhysRevE.53.R4275.
[12] B. S. Kerner, H. Rehborn, Experimental properties of phase transitions in traffic flow, Phys. Rev. Lett. 79 (1997) 4030–4033. doi:10.1103/PhysRevLett.79.4030.
[13] C. Daganzo, M. Cassidy, R. Bertini, Causes and effects of phase transitions in highway traffic, Research report UCB-ITS-RR-97-08, Institute of Transportation Studies, University of California at Berkeley, 1997.
[14] G. Newell, A Theory of Platoon Formation in Tunnel Traffic, Operations Research Society of America, 1959.
[15] J. Hershberger, Finding the upper envelope of n line segments in O(n log n) time, Inf. Process. Lett. 33 (4) (1989) 169–174.