Scheduling vehicles with spatial conflicts

Oddvar Kloster[1], Carlo Mannino[1,2], Atle Riise[1], and Patrick Schittekat[1]

[1]SINTEF Digital, Oslo, Norway, e-mail: atle.riise@sintef.no

[2]University of Oslo, Norway

Abstract

When scheduling the movement of individual vehicles on a traffic network, one must ensure that they never get too close to each other. This is normally modeled by segmenting the network and forbidding two vehicles to occupy the same segment at the same time. This approximation is often insufficient or too restraining. This paper develops and systematizes the use of conflict regions to model spatial proximity constraints. By extending the classical disjunctive programming approach to job-shop scheduling problems, we demonstrate how conflict regions can be exploited to efficiently schedule the collective movements of a set of vehicles; in this case aircraft moving on an airport ground network. We also show how conflict regions can be used in short term control of vehicle speeds, to avoid collisions and deadlocks. The overall approach was implemented in a software system for air traffic management at airports, and successfully evaluated for scheduling and guiding airplanes during an extensive "human in the loop" simulation exercise for Budapest airport. Through simulations, we also provide numerical results to assess the computational efficiency of our scheduling algorithm.

1 Introduction

1.1 Background.

The theoretical and algorithmic results presented in this paper were developed for an application in air traffic management (ATM), namely that of dynamically determining optimal routes and schedules for the collective movement of aircraft and other vehicles on the airport ground surface. This airport surface includes taxiways, runway exits and entrances, holding points, stands, de-icing facilities, and so on. The responsibility for efficient and safe airport ground operations rests with airport ground controllers, who, by means of radio communication with the pilots, dynamically manage the vehicle movements. The focus of this work was to demonstrate how airport ground control can be improved by a combination of dynamic (online) re-planning and automated communication with the aircraft crew to control the taxi route and speed.

The concept is illustrated in Fig. 1. The current situation is continuously evolving as new information arrives (e.g. from the landing of new aircraft, delayed boarding, etc.). Therefore, re-planning is done regularly (e.g. every few seconds) to provide an updated plan for the routing and scheduling of all vehicles. A dedicated *guiding* (or control) software then uses the this plan, together with input about the vehicles' current locations and speeds, to produce a *guiding signal* that shows each pilot where to, and how fast the vehicle should move. In our application, the guiding signal is a combination of adjustable-length Taxiway Center line Lights (TCL) and stop bar lights. However, this guiding signal could also have been communicated by other means, e.g. directly to in-cockpit systems. This automated communication partly

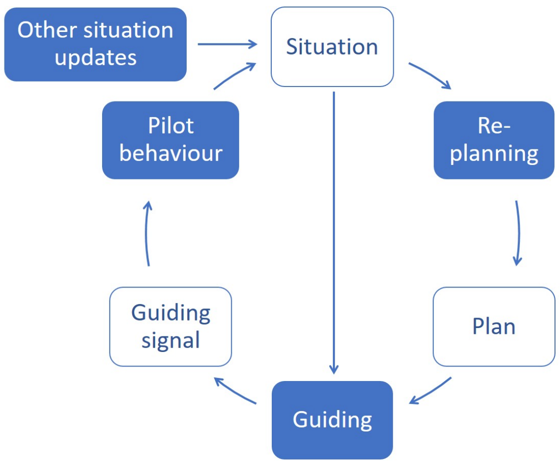

Figure 1: The interaction of the (continuously changing) current situation, continuous re-planning, guiding (control), and pilot behaviour.

replaces radio communication, and thus reduces the work load for the airport ground controllers. Depending on how closely the human pilot follows the guiding signal, the aircraft's movement may deviate - sometimes significantly - from the current plan, which again contributes to the dynamically changing overall situation.

As a suggested replacement of the current manual planning and communication, this concept of continuous integrated re-planning and guiding is new to the airport ground controllers and ATM industry. The development and validation of such new concepts is the purpose of the European modernization program for Air Traffic Management known as the Single European Sky Joint Undertaking (SESAR, [28]). The technology presented in this paper was evaluated as a part of the SESAR project "PJ03a SUMO" ([27]), in collaboration with both airport ground controllers and software industry partners. The resulting software was integrated with an industrial tower control software system and evaluated favourably by airport ground controllers during a one-week validation exercise. The simulation took place at EUROCONTROL research center in Brétigny-sur-Orge, using data from Budapest airport.

1.2 Problem description and literature review

Our dynamic approach depends heavily on the ability to produce realistic adjusted plans for the collective movement of all aircraft in a very short time. To produce the adjusted plan, we solve an optimization problem, in the following called the *Airport Surface Routing and Scheduling Problem* (ASRSP).

The ASRSP may be described informally as follows: We are given a set of vehicles with certain properties in terms of size, speed restrictions, etc. Furthermore, a physical network is defined on which these vehicles could move. There are constraints that may limit the movement of certain vehicles on certain segments of the network, e.g. movement may be illegal, or special speed limits may apply. The problem is to choose a route for each vehicle, and a schedule for the movement of each vehicle along its route, so that no vehicle gets too close to any other. Depending on specific objectives or constraints, different variants of the ASRSP have been addressed in the literature (see, for instance, [4, 12, 13, 16]). In general, we want the schedule to minimize some cost function which is related to an efficient airport throughput, such as punctuality, taxi time from push-back, fuel consumption, etc. In particular, reducing congestion at the airport is a crucial issue (see the thorough discussion of this question in [7]). Solving the ASRSP, besides allowing automatic ground-control operations, represents one viable and powerful approach to achieve this. For example, in an official validation test against expert controllers at the airport of Hamburg (in Fig. 2), the solutions returned by the optimization model led to an average reduction of taxi-time by 35% (see [16]). Solving the ASRSP is an example of a so called *departure metering* approach, which is a class of methods that mitigate congestion through the assignment of appropriate holds for departing aircraft at their gates so as to minimize taxi-out ([5, 22]). In [6], the authors distinguish between two classes of departure metering approaches, namely queue-based and trajectory-based. In the former class, mitigation of congestion is obtained by exploiting queuing and simulation models (as in [5]) or by re-computing (with a given frequency and according to a set of rules) a suitable rate of pushbacks at the gates (e.g. [26]). The ASRSP belongs to the latter class of trajectory-based approaches (as e.g. [12, 13, 16, 21]), which make use of a directed graph to represent the airport surface network and to build the full trajectory for each airplane, from gate to take-off point (and from landing point to gate). Rather than looking only at the time when a departing airplane leaves its gate, these methods actually "follow" the airplane movement from the start (at the gate) to the take-off point, by selecting the spatial route and the time at which the vehicle should be at any point on its route. To this end, the problem is modeled as an optimization problem, which in turn can be solved by various techniques (for instance by Mixed Integer Linear Programming, as in this paper and many others, by Simulated Annealing, as in [6], etc.). Note that since we are solving an optimization model, the target of taxi-out minimization can be pursued explicitly with a term in the objective function. Also, maintaining a high level of runway utilization (another important target in departure metering approaches) is implicitly obtained by considering punctuality as a main component in the objective function. According to the experiments reported in [6], a queue-based approach with robust control performs better than a trajectory-based one, and this is attributed mainly to the uncertain behaviour of airplanes along their taxi-routes. Techniques which explicitly account for uncertainty (as queue-based models and robust control) are more likely to perform better. However, uncertainty can also be handled in optimization models, either directly (as in the robust optimization approach followed in [21]), or indirectly, by simply recomputing an entire plan (i.e. routing and scheduling) for all aircraft whenever a deviation from the current plan occurs, which is done in this paper. Moreover, as also observed in [6], the uncertainty in running times can be mitigated by collecting more and better data or, more drastically, by directly controlling aircraft speed (as suggested in this study).

While we considered the full ASRSP in our validation exercise, the main focus of this paper is on the scheduling subproblem in which the route of each vehicle is taken as given. We call this the *Airport Surface Scheduling Problem* (ASSP).

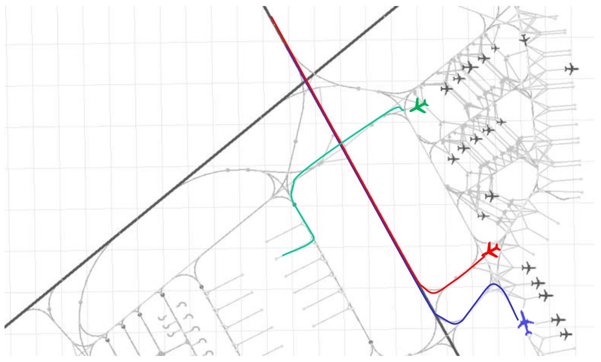
Figure 2: The routes chosen for the blue, red and green aircraft overlap. The schedule must prevent conflicts in the use of shared network segments.

A realistic model of the ASSP requires an accurate representation of the physical proximity between vehicles. Furthermore, it is important that the proximity model can effectively translate into constraints on the variables of the ASSP. The classical approach is to define two vehicles as too close if they are on the same network segment (or neighbouring segments) at the same time. We call this a *resource-based conflict model*, since it is equivalent to treating the problem as a resource constrained scheduling problem [9] where the resources are the network segments. This is also the customary approach to vehicle scheduling with conflict avoidance with severe limitations (see [17]). However, for general network layouts and vehicle geometries, a resource based approach will at best lead to either a model that is too restrictive (introduces unnecessary constraints), or requires a very fine discretisation of the network, compromising the efficiency of the solution process, "wasting non-negligible optimization potential" ([17], Section 5).

To solve the real world ASSP, therefore, we instead model proximity constraints using the concept of *conflict regions*. This model captures exactly where, along their respective routes, two vehicles cannot feasibly be at the same time due to physical (spatial) proximity conflicts. We provide a theoretical background and a concrete algorithm for computing realistic conflict regions based on the vehicles' geometrical properties and the geometry of the network on which they move. Note that a similar concept was introduced independently in [2], [15] and [30], but with some important differences: Where these authors use conflict regions to optimize the movement of vehicles through a local bottleneck (e.g. a junction), we use them for longer term planning of all vehicles on a much larger network (the airport surface). Also, to better tackle general airport layouts and vehicle geometries, we do not limit the shape of conflict regions to hexagons as in [2, 3] or ellipsoids as in [30]. We can do this efficiently because, in contrast to [2, 3], in our approach the shape of the conflict regions does not affect the number of disjunctions (or binary variables) in the optimization model. It is worth observing that in [30] the authors also consider possible random variations in the airborne speed of the aircraft due to wind, a component that we do not address here since only ground operations are involved.

In order to build our optimization model of the ASSP, we first observe that the problem is naturally modelled as a disjunctive program ([8]). Indeed, whenever a potential conflict is identified, for instance when the routes of two aircraft cross at an intersection, the decision of who goes first must be taken. This choice translates into a disjunction between two (sets of) precedence constraints. In the classical resource-based approach, each term in the disjunction is a single linear precedence constraint (see [20]). In our conflict region approach, however, each term may be the conjunction of several linear precedence constraints, determining who goes first through a potentially large set of shared network resources. In Section 5, we present a branch & bound algorithm for the ASSP that exploits this modelling power. Incidentally, note that similar disjunctive programs also arise when modelling and solving airborne conflicts between aircraft (e.g.

[10, 19]).

In addition to being a tool for solving the ASSP, conflict regions are also very useful for the automated guiding described above (see Fig. 1). While the guiding software attempts to guide the aircraft according to the current solution, it must also take into account the fact that human pilots do not always drive at the suggested speeds. To avoid conflicts, the guiding service therefore uses conflict regions to model near future physical proximity, based on current speeds and positions of the all aircraft. The guiding signals (TCL strip lengths and stop bar light signals) are calculated directly based on properties of the conflict regions.

1.3 Contribution

Summarizing, the main contributions of this paper are that:

- we formally introduce conflict regions and related concepts, and prove some relevant properties,
- we describe a procedure for constructing conflict regions,
- we exploit conflict regions in computing optimal schedules by extending the classical disjunctive approach for job-shop scheduling,
- we apply our algorithm to on-line vehicle scheduling, and demonstrate that it is efficient enough for practical use in airport control software,
- we explain how conflict regions can be exploited as a basis for vehicle guiding under non-perfect control,
- we validate our concept of integrated on-line planning and guiding in a large scale human-in-the-loop simulation for a real airport case.

The remainder of this paper is laid out as follows: Sections 2 and 3 show how to represent vehicle movements and conflicts. In Section 4, we show how to model our problem as a disjunctive program. In Section 5, we employ all the novel structures and methods in a branch&bound method for the ASSP as part of a decomposition strategy to solve the full ASRSP problem. Section 6 describes the usefulness of conflict regions in steering the guiding process, while the user evaluation and our numerical experiments are presented in Section 7. The procedures for constructing and extending conflict regions are given in Appendices A and B.

2 Representing conflicts and vehicle schedules

In order to model spatial conflicts we first need a suitable representation of how vehicles move on the physical network. We assume for now that the route for each vehicle is given, and concern ourselves with the timing aspect of a vehicle's movement along this route. Because our experiments deal with vehicles on a surface, we focus here on movements in $\mathbb{R}^2$, but the extension to $\mathbb{R}^3$ is straightforward.

Consider a single vehicle a moving over time along its *route*, R^a, which is a curve in the Euclidean plane. Let L^a be the length of R^a and let $x \in [0, L^a]$ denote the distance the vehicle has moved along its route (e.g. as measured at the nose wheel of an airplane). Note that, for any $x \in [0, L^a]$, R^a determines the vehicle's physical position and orientation at that point in the route.

Let $[0, H]$ be the time horizon and let the timing function, or *vehicle schedule* $t^a(x) \in [0, H]$ define the time at which the vehicle reaches a distance $x \in [0, L^a]$ along its route. A solution to the ASSP consists of the union of all vehicle schedules, denoted by the *schedule* vector $\mathbf{t}$.

In our scheduling model, we approximate each vehicle schedule by a piecewise linear function, t^a , as follows: We first define a set of unique *control points* (coordinates) $p^a_1, p^a_2, \ldots$ on R^a, selected such that the movement of vehicle a can be realistically modelled under the assumption that a's speed is constant between each pair of consecutive points. Typically, such control points are positioned at the beginning of the route and at the intersections of the taxiways (as in [14]), plus possibly at other points in the airport. It follows that each control point p^a_i corresponds to a distance x^a_i along the route at which the speed can change (see Fig.3).

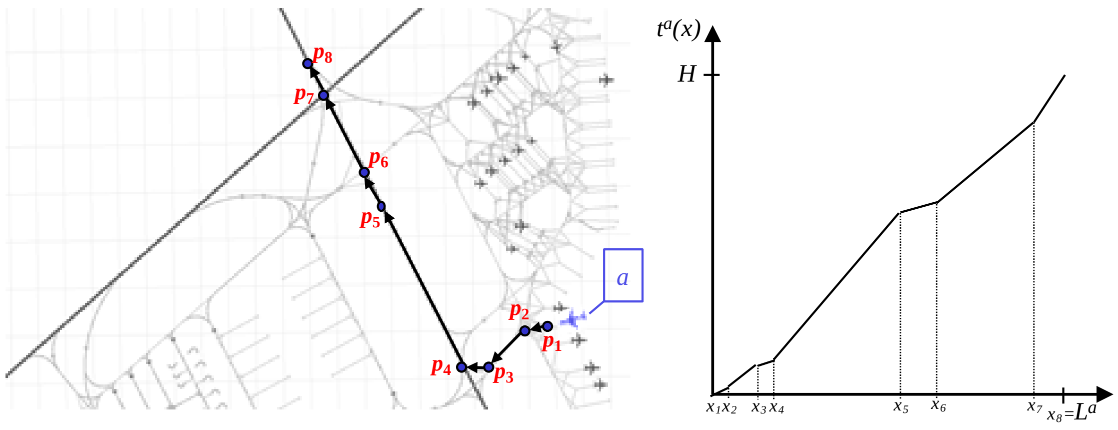

Figure 3: On the left side the route of airplane a across the airport surface with some control points $p_1, p_2, \ldots$. On the right, the corresponding trajectory, assuming constant speed between successive positions $x_1, x_2, \cdots \in [0, L^a]$.

It follows that the $t^a(x)$ is piecewise linear, with breakpoints only at the distances $x_1^a, x_2^a, \cdots \in [0, L^a]$. Also, $t^a(x)$ is continuous, since our definition of $t^a(x)$ does not allow a vehicle to stop at a single point (exception at the boundary, where $x^a = L^a$). As a scheduling model, this is sufficient, as we can always add control points very close to each other on the route if it is necessary to model (almost) stopping. However, during the real execution of the plan, the vehicles will normally deviate from the constant speed assumption. In Section 6 we will explain how, in our application, automated guiding plays together with the on-line scheduling, to tackle this.

For the following discussion, we also introduce the *position function* $x^a(t)$, which defines the vehicle's position (or distance) along its route at any time $t \in [0, H]$. Note that $x^a(t)$ is continuous and monotonically increasing, with $x^a(0) = 0$ and $x^a(H) = L^a$. Under our "constant speed, no stopping" assumption above, $x^a(t)$ is the inverse function of $t^a(x)$, and is also piecewise linear.

2.1 Conflict regions

Consider now the concurrent movements of a pair of vehicles, a and b, along their routes R^a and R^b of length L^a and L^b, respectively. For any $t \in [0, H]$, a will be at $x^a(t)$ and b

will be at $x^b(t)$. We call the pair $(x^a(t), x^b(t))$ the *concurrent position* of a and b at time t. We further define the $\mathbb{R}^2$ region of all such concurrent positions, $B_{ab} = \{(x^a, x^b) : x^a \in [0, L^a], x^b \in [0, L^b]\}$, as the *concurrent positions space*.

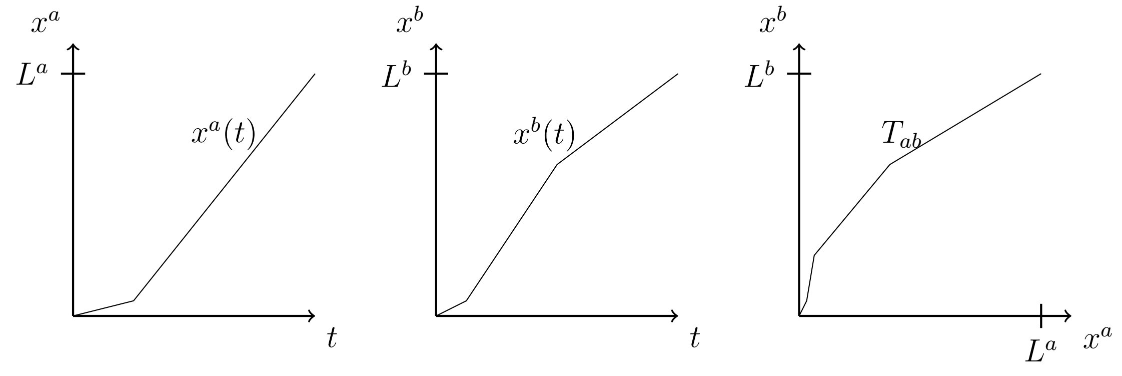

Figure 4: Two position functions and the concurrent trajectory

Suppose now that the situation corresponding to $(x^a, x^b) \in B_{ab}$ is impossible, due to the vehicles occupying a common region of physical space, or undesirable, due to the vehicles being too close for safety. We then call the point (x^a, x^b) a *conflict point*. A *conflict region* is a connected set $C \subseteq B_{ab}$ of conflict points. The exact shape of C in the B_{ab} plane depends on the geometrical shape of each vehicle, the geometrical properties of the network on which the vehicles move, and the vehicles' routes on this network. Note that there can be several conflict regions in B_{ab}. A theoretical basis and a concrete algorithm for constructing conflict regions are given in Appendix A.

2.2 Concurrent trajectories

The set of concurrent positions that are actually realized, with a choice of position functions $x^a(t)$ and $x^b(t)$, is $T_{ab} = \{(x^a(t), x^b(t)) : t \in [0, H]\}$.

We call T_{ab} the *concurrent trajectory* of a and b. While $T_{ab} \subset B_{ab}$ is technically a set, we can also consider it as a continuous parametrized curve $f(x^a, x^b) = 0$. The *concurrent trajectory* T_{ab} has the following important property:

Property 2.1 (Monotonicity) *Let $(x_0^a, x_0^b), (x_1^a, x_1^b) \in T_{ab}$. (i) If $x_1^a > x_0^a$, then $x_1^b \geq x_0^b$. (ii) If $x_1^b > x_0^b$, then $x_1^a \geq x_0^a$*

Proof. We show (i) ((ii) is analogous). For $i = 0, 1$, let $t_i \in [0, H]$ be such that $x_i^a = x^a(t_i)$ and $x_i^b = x^b(t_i)$. Since $x_1^a > x_0^a$ and $x^a(t)$ is non-decreasing, we have $t_1 > t_0$. Then, since $x^b(t)$ is non-decreasing, we have $x_1^b = x^b(t_1) \geq x^b(t_0) = x_0^b$. □

Recall that x^a and x^b are piecewise linear functions of t. We can therefore partition the domain $[0, H]$ into intervals such that in each interval, both x^a and x^b are linear functions of t (and not just piecewise linear). Consequently, T_{ab} describes a line segment in B_{ab} when restricted to one of these intervals, and we have the following:

Property 2.2 *If both $x^a(t)$ and $x^b(t)$ are piecewise linear functions in $[0, H]$, then the concurrent trajectory is also piecewise linear.*

2.3 Trajectory feasibility

Consider the concurrent positions space B for vehicles a and b, with a single conflict region C (in the following, we skip the subscript when denoting the concurrent positions space B_{ab} and the concurrent trajectory T_{ab}). We say that a (concurrent) trajectory T is *feasible* if it does not pass through the interior of the conflict region C, and *infeasible* otherwise (see Fig. 5).

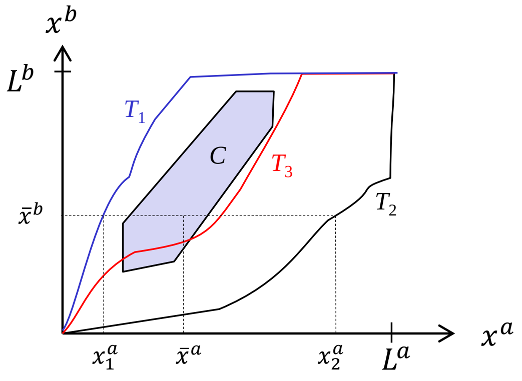

Figure 5: A conflict region C. The concurrent trajectories T_1, T_2 are feasible, whereas T_3 is infeasible. T_2 is below-right, T_1 is above-left. Note that $(\bar{x}^a, \bar{x}^b) \in C^\circ$, $(x_1^a, \bar{x}^b) \in T_1$ and $(x_2^a, \bar{x}^b) \in T_2$, with $x_1^a < \bar{x}^a < x_2^a$
.

In general, a conflict region can be difficult to describe analytically. We will assume that the conflict region is described by a polygon, which is sufficient in our practical application. Furthermore, we will assume C is open in B, i.e. a concurrent trajectory T is infeasible if it intersects the interior C° of the conflict region C. In Appendix B we will show that there may be concurrent positions outside the conflict region, which cannot possibly belong to any feasible concurrent trajectory because any concurrent trajectory through such a point must also intersect C. In the following, unless otherwise specified, we consider conflict regions that have been "extended" to include all such points.

Above or below. Given a conflict region C, we can classify the feasible concurrent trajectories according to their position with respect to C. Informally, a feasible trajectory either lies above and left of C°, or it lies below and right of C°. In the first case we say that vehicle b goes first, because b traverses the positions corresponding to potential conflict points before vehicle a. In the other case, we say that a goes first.

Remark 2.3 *Let T be a feasible trajectory, and C be an (extended) conflict region. Then we have two possibilities:*

1. *(b goes first): the trajectory lies above and left to the conflict region. That is, for any $\bar{x}^b \in [0, L^b]$ we have that $(x_T^a, \bar{x}^b) \in T$ and $(x_C^a, \bar{x}^b) \in C^\circ$ implies $x_T^a < x_C^a$. Similarly, for any $\bar{x}^a \in [0, L^a]$ we have that $(\bar{x}^a, x_T^b) \in T$ and $(\bar{x}^a, x_C^b) \in C^\circ$ implies $x_C^b < x_T^b$.*

2. *(a goes first): the trajectory lies below and right to the conflict region. That is, for any $\bar{x}^b \in [0, L^b]$ we have that $(x_T^a, \bar{x}^b) \in T$ and $(x_C^a, \bar{x}^b) \in C^\circ$ implies $x_C^a < x_T^a$. For any $\bar{x}^a \in [0, L^a]$ we have that $(\bar{x}^a, x_T^b) \in T$ and $(\bar{x}^a, x_C^b) \in C^\circ$ implies $x_T^b < x_C^b$.*

So, if b goes first, any point on the trajectory lies to the left of any infeasible point at same height and above any infeasible point with same x-coordinate. We call such a trajectory an *above-trajectory*, ($\uparrow$) with respect to C. If a goes first, any point on the trajectory lies to the right of any infeasible point at same height and below any infeasible point with same x-coordinate and we call the trajectory a *below-trajectory* ($\downarrow$) with respect to C.

The above Remark 2.3 has very important consequences for the construction of concurrent trajectories.

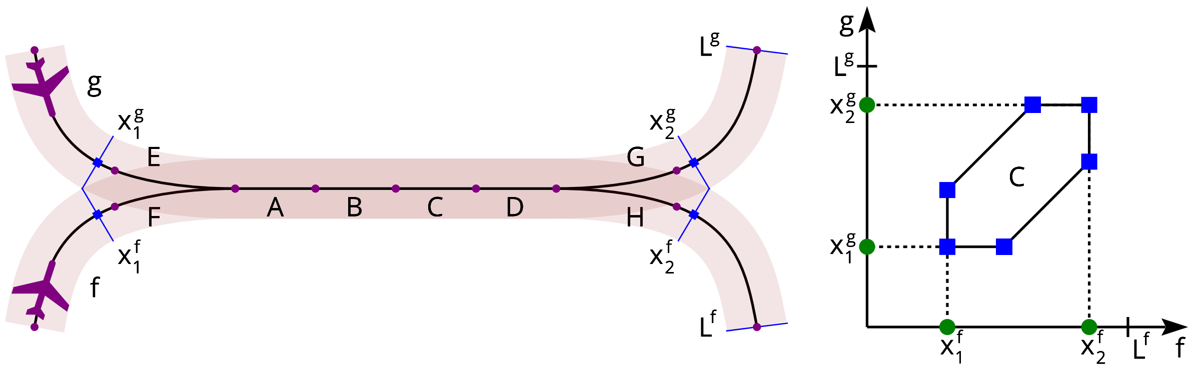

Figure 6: Two airplanes sharing the same path. To the left, the paths on the airport surface, and the shaded areas where the aircraft may be in conflict. To the right, the corresponding conflict region, C.

2.4 Modelling power

In the resource based approach for traffic management problems (e.g. [20, 25]), the physical routes are "discretized" into fixed capacity segments, typically corresponding to some physical architecture elements. For instance, railway tracks are subdivided into block sections, which in practice are always preceded by a physical signal (as a traffic light): each block section can accommodate at most one train. Similarly, airport taxiways are subdivided into unit capacity segments, where each segment must be cleared before a vehicle can enter it. To ensure this, the timing variables are associated with the beginning of each physical segment, and disjunctive constraints ensure that, for each segment shared by two vehicles, one vehicle can enter only when the other one has left the segment.

This resource based approach has some severe limitations in terms of modelling power. First, unless all segments are quite short, the disjunctive constraints will be too restrictive. Second, disallowing the sharing of single segments is not enough, since two aircraft can be of two connected segments and still collide because they are both close to the connecting node. Next, consider the small example in Fig. 6. Here, the two aircraft f and g have a shared path along the segments (A, B, C, D). It is obvious from the left hand figure that if f goes first on A, it must also go first on B, C, and D, since the airplanes cannot overtake each other on their common path. However, the resource based approach offers no immediate way of expressing this. You will need to decide who goes first for each segment individually. This applies whether the aircraft move in the same, or in opposite directions on their common path. Furthermore, there are many situations where two aircraft are moving on different network segments, but still are close enough to each other to be in physical conflict. One example is the movement of f and g on segments E and F, respectively, in Fig. 6. The resource based approach offers no immediate way of representing this. It is of course possible to work around these limitations, even within the resource based model. However, this requires extensive ad-hoc analysis of each case, and the introduction of a large number of extra artificial resources, and/or extra constraints. Also, this analysis would have to carefully consider the geometry of the aircraft and the movement network.

The conflict region approach offers a generic way of performing this spatial analysis, and a rather more elegant alternative way of modelling the spatial conflicts. For example, one would model all the proximity constraints of Fig.6 (including proximity of aircraft on segment pairs E, F and G, H) with a single conflict region C. This means that there is only one choice of who goes first for the the entire conflict: should the concurrent trajectory be above or below C. Also, the exact distances at which the conflict begins or ends are known, independently of the segment definition (see the right hand side of Fig .6). The same generic logic applies regardless of the direction of movement of the aircraft, or of whether they use the same network segments or merely segments that are close to each other. The only difference would be the shape of C.

3 Constructing feasible concurrent trajectories.

In the following we show how we can construct a concurrent below-trajectory ($\downarrow$) with respect to a single conflict region C, as well as the constraints on the problem variables that this concurrent trajectory must satisfy in order to be feasible (the logic for above-trajectories is analogous). We start from our assumption that the vehicle schedules $t^a(x^a)$ and $t^b(x^b)$ are piecewise linear. Let $G^a = \{0 = x^a_1, x^a_2 \dots, x^a_q = L^a\}$, where $x^a_1 < x^a_2 \dots < x^a_q$ denote the distances along the route of a corresponding to potential breakpoints of

$t^a(x^a)$, namely points on the route where the speed of a can change (in Section 7.1 we describe how such points are selected for our real-life cases). This implies that $t^a(x^a)$ is completely defined by its values in these breakpoints, and we take as our timing variables $t^a_1 = t^a(x^a_1), t^a_2 = t^a(x^a_2), \dots t^a_q = t^a(x^a_q)$. Note that the minimum time for a to move from x^a_i to x^a_{i+1} is an input parameter λ^a_i, and so the timing variables must respect the following set of speed constraints:

$$t^a_{i+1} - t^a_i \geq \lambda^a_i \qquad i = 1, \dots, q-1 \tag{1}$$

Similarly, for vehicle b, we define $G^b = \{0 = x^b_1, x^b_2 \dots, x^b_r = L^b\}$, with $x^b_1 < x^b_2 < \dots < x^b_r$, and the set of timing variables $t^b_1, t^b_2, \dots t^b_r$, where t^b_j is the time when b reaches $x^b_j \in G^b$.

We now define a rectangular grid G over the concurrent positions space B, by the points $G = \{(x^a_i, x^b_j) : x^a_i \in G^a, x^b_j \in G^b\}$, as depicted in Fig. 7.a). By Property 2.2, any concurrent trajectory is piecewise linear and can only have breakpoints in G.

We start by building a *reference* below-trajectory $T^{\downarrow}_C$ such that no other feasible concurrent below-trajectory can go above $T^{\downarrow}_C$. In other words we build $T^{\downarrow}_C$ "as close as possible" to the conflict region.

$T^{\downarrow}_C$ can be described by a finite ordered list $Q^{\downarrow}_C = ((0,0) = p_1, p_2, \dots, (L^a, L^b))$ of breakpoints in G. To be feasible, we need to make sure that, for any two successive points $p_j, p_{j+1} \in Q^{\downarrow}_C$, the segment joining p_j and p_{j+1} does not intersect the interior of the conflict region. To construct $T^{\downarrow}_C$, first observe that the grid G subdivides region B into rectangles (Fig. 7). Each such rectangle R_{ij}, for $i = 1, \dots, q-1$, $j = 1, \dots, r-1$, is identified by the set of corner points $\{(x^a_i, x^b_j), (x^a_{i+1}, x^b_j), (x^a_{i+1}, x^b_{j+1}), (x^a_i, x^b_{j+1})\}$. The diagonal from the left-lower corner to the right-upper corner splits a rectangle into two triangular regions, the *upper triangle* and the *lower triangle* (Fig. 7). Consider now the set $\mathcal{R}^{\downarrow}$ of rectangles of the grid that contain some conflict points, but such that there are no rectangles below which contain conflict points; these are coloured in light blue in Fig. 7.a). The set of breakpoints of $T^{\downarrow}_C$ will contain only corners of such rectangles. In particular, for each rectangle $R \in \mathcal{R}^{\downarrow}$, we distinguish two cases. If there are conflict

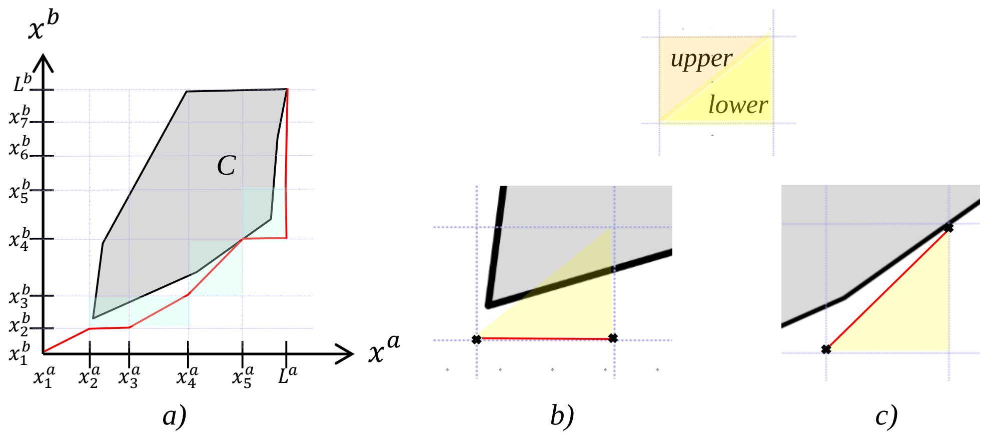

Figure 7: The grid. The reference trajectory in red goes below C passing through corners of the grid rectangles that contain the lower boundary. In b) and c) we show how such passing corners are chosen.

points in the lower triangular region of R, then the set of breakpoints contains the two bottom corners of R (Fig. 7.b)). Otherwise, all conflict points lie in the upper triangle of R and the set of breakpoints contains the bottom-left corner and the top-right corner of R (Fig. 7.c)). Finally, $(0, 0)$ is the first point of the trajectory, and (L^a, L^b) is the last. By construction, the concurrent trajectory so built lies entirely below the conflict region and is as close as possible to it (for the given set of potential breakpoints). The reference concurrent above trajectory $T_C^{\uparrow}$ is constructed in an analogous way.

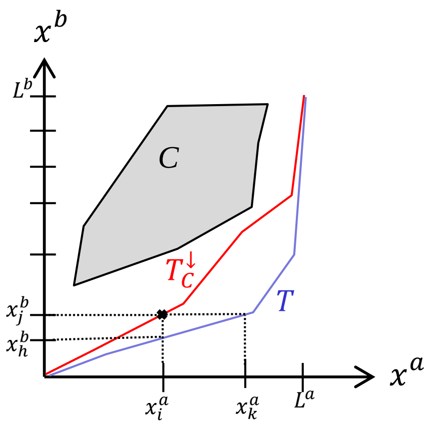

Figure 8: The reference concurrent below-trajectory $T_C^{\downarrow}$ and another concurrent trajectory T which lies below it. On the reference trajectory, vehicles a and b arrive simultaneously at point (x_i^a, x_j^b). On trajectory T, instead, when a is in x_i^a, b is in $x_h^b < x_j^b$. When b reaches x_j^b, a is in $x_k^a > x_i^a$.

Constraints for feasible concurrent trajectories. If $T_C^{\downarrow}$ is the reference below-trajectory (with breakpoint list $Q_C^{\downarrow}$), and the actual trajectory T lies at or below $T_C^{\downarrow}$, then, for any breakpoint (x_i^a, x_j^b) of $T_C^{\downarrow}$, at the time a reaches x_i^a, b has not yet passed x_j^b (see Fig. 8). This occurs if and only

$$t_i^a \leq t_j^b \quad (x_i^a, x_j^b) \in Q_C^{\downarrow} \tag{2}$$

Symmetrically, if the actual trajectory T lies at or above the reference above-trajectory $T_C^{\uparrow}$ is the reference above-trajectory (with breakpoint list $Q_C^{\uparrow}$), then we have:

$$t_i^a \geq t_j^b \quad (x_i^a, x_j^b) \in Q_C^{\uparrow} \tag{3}$$

Since any feasible concurrent trajectory is either a below-trajectory or an above-trajectory (w.r.t. C), then any feasible schedule $\mathbf{t}$ satisfies either all constraints (2) or all constraints (3). The pair of constraint systems (2) and (3) forms a *disjunctive constraint system pair.*

4 Computing Feasible schedules

The previous discussion is limited to two vehicles and one conflict region. In general, we have a fleet F of vehicles. For each $a \in F$ we are given its route and the set $G^a = \{x_1^a, x_2^a, \dots\}$ of distances along the route where the vehicle can change its speed. Next, for each pair $\{a, b\} \subseteq F$ of vehicles, we may have several disjoint conflict regions $\mathcal{C}^{ab}$. Our problem is to find a (feasible) schedule for each vehicle so that some cost function $c(\mathbf{t})$ of the overall schedule is minimized. To this end, we proceed as follows:

1. For each pair $\{a, b\} \subseteq F$ of vehicles compute the set of disjoint conflict regions $\mathcal{C}^{ab}$
2. For each pair $\{a, b\} \subseteq F$ and each conflict region $C \in \mathcal{C}^{ab}$ compute the two reference trajectories and the alternative lists of breakpoints $Q_C^{\uparrow}$ and $Q_C^{\downarrow}$.

3. Solve the following disjunctive program

$$
\begin{array}{ll}
\min c(\mathbf{t}) & \\
(i) \quad t^a_{i+1} - t^a_i \geq \lambda^a_i, \quad x^a_i \in G^a \setminus \{L^a\}, a \in F & \\
(ii) \quad \left.\begin{array}{ll} t^b_j - \ t^a_i \geq 0 & (x^a_i, x^b_j) \in Q^{\downarrow}_C, \\ \bigvee & \\ t^a_i - \ t^b_j \geq 0 & (x^a_i, x^b_j) \in Q^{\uparrow}_C \end{array}\right\} C \in \mathcal{C}^{ab}, \{a,b\} \subseteq F & \\
t^a \in \mathbb{R}^{G^a}_+, a \in F &
\end{array} \tag{4}
$$

Constraints (4.i) are the speed constraints (1). We also have other fixed time precedence constraints corresponding to time windows, but we drop them here for sake of simplicity. Constraints (4.ii) are the disjunctive systems of constraints introduced in the previous section. If, for a given pair of vehicles and for a conflict region C, the schedule satisfies the inequalities in the first term of (4.ii), then the trajectory will go below C. Otherwise, the schedule must satisfy the second term and the trajectory will go above C.

Program (4) extends the disjunctive model for job-shop scheduling with blocking, no wait constraints discussed by Mascis and Pacciarelli in [20], which in turn extends the seminal model for job-shop scheduling introduced by Balas in [8]. In these classic models, each disjunctive constraint is the disjunction of two terms. Each term is a time precedence constraint of the form $t_j - t_i \geq l_{ij}$. For every disjunctive constraint, exactly one of the two terms must be satisfied by any feasible schedule $\mathbf{t}$. In our model, the disjunctive constraint (4.ii) is again the disjunction of two terms. But now each term is the conjunction of several time precedence constraints, one for each breakpoint of a list. Already in its simple form with only one time precedence constraint per term, solving Program (4) is NP-hard (see [20]), even when the objective function $c(\mathbf{t})$ is linear.

Observe that choosing "who goes first" for all conflicts $C \in \mathcal{C}^{ab}$, for all vehicle pairs $\{a, b\} \subseteq F$, corresponds to choosing which term in each disjunction must be satisfied by $\mathbf{t}$, or, equivalently, to dropping one term for each disjunction from Program (4).

This leaves a linear program later denoted as *timing problem.* The timing problem can be shown (see, for instance, [18]) to be the dual of a min-cost flow problem and can be solved effectively by ad-hoc algorithms, such as the Network Simplex Method (see [1]).

Objective function. The objective function is a linear combination of different cost components. Some of these concern timing variables, depending on vehicle schedule makespans, deviations from target times for lineup or parking, and deviations from preferred speeds of vehicles. Another objective component prioritises, in any conflict resolution, a vehicle that was cleared for taxi over any vehicle that had not yet received such clearance. Finally, in order to maintain some stability in the proposed solutions, an objective was defined to penalise changes in sequencing priorities as compared to the currently employed solution. That is, the solution is penalised for each spatial conflict where the choice of "who goes first" is opposite to the corresponding choice in the currently accepted plan. The presence of this objective was critical to the evaluation of the methodology, as it let the model react adequately to changes to the situation while still not bothering the user with wildly fluctuating sequencing decisions.

To provide a formal description of our objective function $c(t)$ we need to introduce a number of support variables. First, for each $C \in \mathcal{C}^{ab}$, $\{a, b\} \subseteq F$, we introduce variable y_C^{ab}, with $y_C^{ab} = 1$ if and only if a goes first. We then define k_C^{ab} to be the cost of letting a go before b in a conflict C (the wanted order has zero cost). Next, for each $a \in F$, depending on whether a is departing (lining-up) or arriving (parking) let T_p^a be the target time and t_p^a be the time variable associated with the line-up point or parking point of a. Then, we let $d_p^a = \max(0, t_p^a - T_p^a)$ be the delay and $e_p^a = \max(0, T_p^a - t_p^a)$ be the earliness (at line-up or at parking depending on the nature of a), with c_p^a be the cost of a unit of delay or of earliness. Next, a preferred running time γ_i is given for any pair of successive break-points x_i^a, x_{i+1}^a on the trajectory of $a \in F$. Then we let $s_i^a = |t_{i+1}^a - t_i^a - \gamma_i|$ be the running time deviation and c_s^a be the cost of a unit of deviation.

Then the objective function $c(t)$ is the sum of the following terms:

- *Earliness-Tardiness:* $\sum_{a \in F_D} c_p^a d_p^a + \sum_{a \in F_A} c_p^a e_p^a$, where F_D is the set of departures, and F_A is the set of arrivals.

- *Running time deviation:* $\sum_{a \in F} \sum_{x_i^a, x_{i+1}^a \in G^a} c_s^a s_i^a$.
- *Priority:* $\sum_{\{a,b\} \subseteq F} \sum_{C \in \mathcal{C}^{ab}} k_C^{ab} y_C^{ab}$.
- *Stability:* Let us denote by $\bar{\mathbf{y}}$ the ordering of the flights associated with the potential conflicts in the currently accepted solution. Then the cost of "changing" the solution will be $c_s|\mathbf{y} - \bar{\mathbf{y}}|$, where c_s is the cost of one exchange.

5 Solution algorithm

We are now ready to describe our solution algorithm for ASRSP. We consider first the ASSP sub-problem.

5.1 Solving the scheduling problem

In the ASSP, the set of routes R is pre-determined, one route for each vehicle. The ASSP is completely expressed by Program (4), and its solution is an optimal schedule $\mathbf{t}$ as defined in section 2. Conflict regions between each pair of aircraft are also in input to our problem. Each region is defined completely by the (sub-) routes and the physical sizes of the involved aircraft. It needs to be computed at most once (using the efficient procedure described in the appendix), and can then be saved for later use. In practice, therefore, the computation of the conflict diagrams has no impact on the solution time.

We solve the ASSP by Branch & Bound with *delayed constraint generation* ([23]), i.e. we iteratively generate the constraints of (4). The algorithm works as follows: We start by dropping all disjunctive constraints ($4.ii$) and by keeping only constraints ($4.i$). We denote this linear program by P_0. Note that, since the objective function is linear and all constraints are of the form $t_y - t_x \geq l_{xy}$ (plus upper and lower bounds on the variables), problem P_0 is the dual of a min-cost flow problem. We solve P_0 by the network simplex method to find an optimal schedule $\mathbf{t^0}$ (if no such schedule exists then the overall problem is infeasible). If, by chance, $\mathbf{t^0}$ does not violate any proximity constraints, then $\mathbf{t^0}$ is also feasible and optimal for the overall problem (4) and we are done. However, because P_0 is a relaxation of the original problem, $\mathbf{t^0}$ may be infeasible for the ASSP. In this case, there is at least one pair of vehicle $\{a, b\} \subseteq F$ whose concurrent trajectory T_{ab} intersects the interior of a conflict region $C \in \mathcal{C}^{ab}$ and the corresponding disjunctive constraint is violated by $\mathbf{t^0}$. According to the row generation approach, we add this constraint to the problem and solve the resulting non-linear program. In practice, we immediately branch on the disjunctive constraint by creating two new *branching node problems* $P_0^{\uparrow}$ and $P_0^{\downarrow}$, corresponding to the cases of the trajectory T_{ab} going above or below C, respectively. Each of these are constructed by extending P_0 with the corresponding term in the disjunction, namely by adding a set of time precedence constraints. Note that each new problem is again a timing problem, i.e. the dual of a min-cost flow problem, and can be solved in the same fashion as P_0. The process is thus repeated recursively, backtracking whenever a timing problem is infeasible, when we can cut on objective bounds, or when a new best feasible (i.e. without conflicts) solution is found.

The scheduling algorithm is illustrated by pseudocode (5.1) where $\mathcal{T}$ is the list of open problems to solve, whereas $\bar{\mathbf{t}}$ is the incumbent best solution. Bounding is performed in the standard way. Initially $\mathcal{T} = \{P_0\}$ and the incumbent solution $\bar{\mathbf{t}}$ may be found by some heuristic or, if no incumbent is available, we let $c(\bar{t}) = \infty$.

Algorithm 5.1: SurfaceScheduling(R)

```
$\mathcal{T} \leftarrow \{P_0\}, \bar{\mathbf{t}}$
while $\mathcal{T} \neq \emptyset$
  do
    Pick a problem $P$ from $\mathcal{T}$ and let $\mathcal{T} \leftarrow \mathcal{T} \setminus \{P\}$
    Solve $P$ and let $\mathbf{t}^*$ be the solution.
    if $P$ is feasible and $c(\mathbf{t}^*) < c(\bar{\mathbf{t}})$
      then
        if $\mathbf{t}^*$ is globally feasible (no conflicts are violated)
          then $\bar{\mathbf{t}} \leftarrow \mathbf{t}^*$
          else
            Select a violated conflict region $C$
            Generate two new sub-problems $P^{\uparrow}$ and $P^{\downarrow}$
            $\mathcal{T} \leftarrow \mathcal{T} \cup \{P^{\uparrow}, P^{\downarrow}\}$
return $(\bar{\mathbf{t}})$
```

As mentioned before, the problem solved at each branching node is the dual of a min-cost flow problem. Our implementation of the simplex network algorithm exploits the fact that each node problem is obtained from its parent by only adding a few linear inequalities, and the previous solution can be used to warm start the next optimization run. This has a significant impact on the performance of our scheduling algorithm.

Note that our approach is somewhat similar to the resource-based branch and bound search of [29], in that at each node a timing sub-problem is solved before checking for conflicts, and that each *conflict resolution* consists of a set of linear time precedence constraints. In our case, however, exploiting conflict regions lets us represent each conflict with a single disjunctive constraint (4.*ii*), rather than one disjunctive constraint per resource. We thus get a much shallower search tree, and so a much more efficient algorithm. Note that this gain in efficiency is not limited to tree search algorithms. One can for example think of local search algorithms that exploits conflict regions to efficiently model "who goes first" decisions.

5.2 Solving the full routing and scheduling problem

Now, to solve the full ASRSP, we need to also consider the routing aspect. Many airports, including the airport that we considered in our project, operate with the concept of *default taxi routes*, which is a set of pre-defined routes between each pair of possible start and destination points. Default routes may be different for different types of aircraft or other vehicles.

Other routes are chosen only when we have a deadlock. This can happen, for example, when pilots do not comply strictly with the current plan (see also Section 6), causing the current position of a pair of vehicles to be in the extended part of the conflict region. Also, there may be a *gridlock* in which the current position of three or more vehicles together makes the scheduling problem infeasible. In such cases, the algorithm must find alternative route(s) for one or more vehicles, and a new optimal schedule must be computed. The overall algorithm is illustrated in Fig. 5.2.

Algorithm 5.2: SURFACE ROUTING AND SCHEDULING()

```
$R \leftarrow$the default route for each vehicle

do
    $\mathbf{t} \leftarrow SurfaceScheduling(R)$   (see Algorithm 5.1)
    if feasible
      then return $(R, \mathbf{t})$
      else $R \leftarrow$Resolve deadlocks or gridlocks by rerouting some vehicle(s)
```

The re-routing is done in a heuristic fashion, using Dijkstra's shortest path algorithm with some special requirements to avoid certain points in the network (and thus resolve the deadlock). As our re-routing procedure is rather straightforward, and not the focus of this paper, we omit the details of this.

6 Guiding

In our application the "current plan" is continuously adjusted by on-line re-solving of the ASRSP. A dedicated *guiding system* then communicates this plan to each pilot, providing information about the aircraft's planned route and speed, and the need to yield to other vehicles (see Fig. 1). Since the aircraft are piloted by humans, however, their actual speed will often deviate from the plan. We therefore need the guiding logic to act as a safety net, to makes sure that the aircraft follows the current plan with sufficient accuracy. In particular, the guiding ensures the correct sequencing of aircraft at any junction or lineup, and prevents aircraft to get too close to each other or to enter a deadlock situation.

The guiding communicates speed by means of a sequence (or "*strip*") of green center-line lights where the length of the light strip indicates speed. The logic is that the front of the aircraft should be at the end of the light strip in a certain number of seconds into the future. We believe this is an intuitively understandable way to communicate speed adjustments; A "standard" strip length indicates that the current speed should be maintained, an increasing strip length indicates that the aircraft should speed up, and a decreasing strip length indicates that the aircraft should slow down. An aircraft should never pass the end of the light strip (one may also signal a full stop by letting the last lamp on the strip be red instead of green).

As long as an aircraft taxis follows it's schedule exactly, it is easy to calculate the strip length from the current plan. As mentioned above, however, this is rarely the case, and the strip length calculations must also take the *actual* position and speed into account. We find that conflict regions still offer an elegant way of computing the correct light strip length. Consider the situation in Fig. 9.

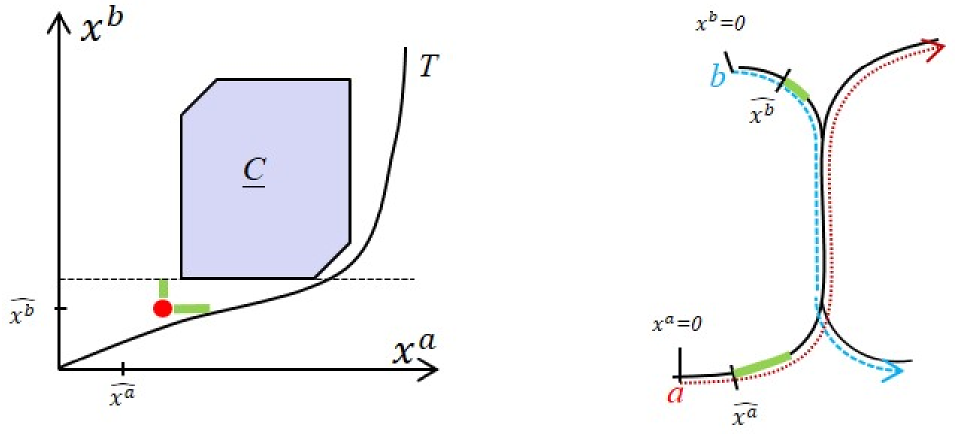

Figure 9: A situation where the concurrent position of vehicles a and b has deviated from the planned concurrent trajectory T, approaching the conflict region. The relative speed of b to a needs to be reduced because the concurrent position is above T.

As it can be seen from the planned concurrent trajectory T in the left part of the figure, we want a to go before b on the overlapping part of their respective routes. However, their relative movement has deviated from T, because b has gone too fast, a too slow, or both. To tell b to yield to a, the light strip of b must end before b goes too far. Indeed, the last possible stopping point corresponds to the position where b reaches the lowest point of the conflict region; if b goes further, a can no longer go first. We can therefore calculate the length of the light strip directly as the distance between the current position of b along its route (i.e. $\widehat{x^b}$ in Fig. 1) and the conflict region's projection on the x^b axis.

7 Performance and Real-life Validation

The methodology presented in the paper was validated in a project under the umbrella of EU's SESAR Joint Undertaking [11], in which new technologies, concepts and modes of operations for ATM are developed and validated. In our project, the main new concepts were the continuous re-scheduling aircraft movements, coupled with an above ground level-based guiding of vehicles that also included using the length of the light strip to indicate speed. Furthermore, an automatic detection of up-coming deadlock- and gridlock situations were to be communicated not only as alarms, but accompanied with a suggested re-routing of the involved vehicles. The controllers could then just accept this re-routing, as opposed to having to manually come up with re-routing solutions.

7.1 Experiments and numerical results

The validated methodology depended on the ability of our solver to compute optimal adjustments to the current plan every few seconds. It is therefore interesting to assess the performance of our optimization methods. For this we used the data sets that was made available by Budapest Airport, and compiled by Eurocontrol, for the purpose of the technology evaluation mentioned above. These comprised a total of five realistic scenarioes, each of a duration of 1 to 1.5 hours, starting in the early morning. Using real time simulation, we ran through all of these scenarioes, and solved a new problem instance every few seconds to continuously update the current plan. Each such problem instance was therefore quite similar to the previous one, except that some time had passed during which new arriving or departing flights may have been added, some taxi movements may have been completed, and the (simulated) pilots and controllers may have deviated from the previously accepted schedule for push-back and taxi of each individual aircraft. In total, 11457 optimization problem instances were generated during the five scenario simulations.

Our focus here is the scheduling subproblem, which is the computationally challenging part of the full algorithm in 5.2. We applied the scheduling algorithm 5.1 to solve the disjunctive program (4) for each problem instance. A small fraction of these (3.4%) could not be solved, because the simulation, without human input, sometimes generated gridlocks making the scheduling problem infeasible. In a real world settings, these deadlocks would be reported back to the controller with suggested re-routing fixes; in our experiments we simply ignored these instances. All the remaining instances were solved to optimality.

Recall that timing variables must be defined for a set of distances G^a along the route of each aircraft, a. In our implementation, these timing function breakpoints were defined as follows: For each aircraft a, G^a is initially based on the topological network nodes along the route of a (see Section 3). In some situations, e.g. due to (simulated) pilots going faster than indicated by the current schedule, these breakpoint sets lack the necessary flexibility to make the scheduling problem feasible. I.e., the current concurrent position of a pair of aircraft may be too close to the (extended) conflict region, so that any schedule based on their basic breakpoint sets will be infeasible. In this case, we added additional breakpoints between the aircraft's current position and the position where the aircraft would enter the extended conflict region. Note, however, that even in such situations, the automated guiding always ensured that no pair of aircraft actually enter an extended conflict region (see Fig. 9).

The experiments were carried out on a normal laptop computer, an HP ZBook 15 G4, 32 GB RAM, 4-core Intel(R) Core(TM) i7-7700HQ cpu.

Fig. 10 shows the distribution of problem instances across problem sizes (number of aircraft). Fig. 11 illustrates the average run times for our algorithm, within each of these size classes. It shows that for problems of up to 27 aircraft, the average computation time is mostly within 100 milliseconds, and all run times were well below 200 milliseconds.

We conclude that at least for problem instances of these sizes, the performance of the scheduling algorithm is sufficient for real life use.

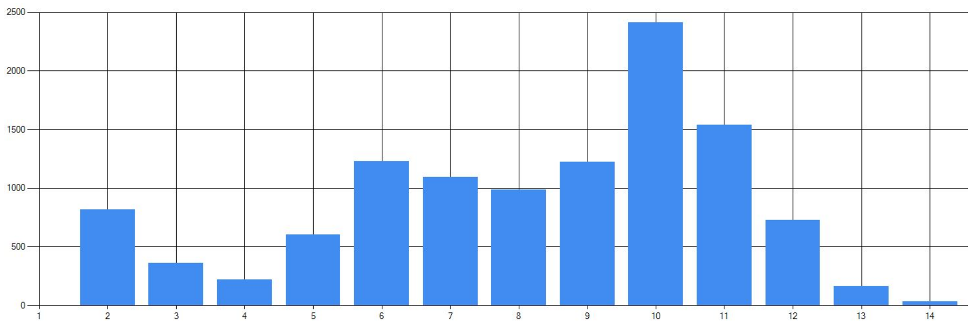

Figure 10: The number of problem instances for different problem sizes (number of aircraft).

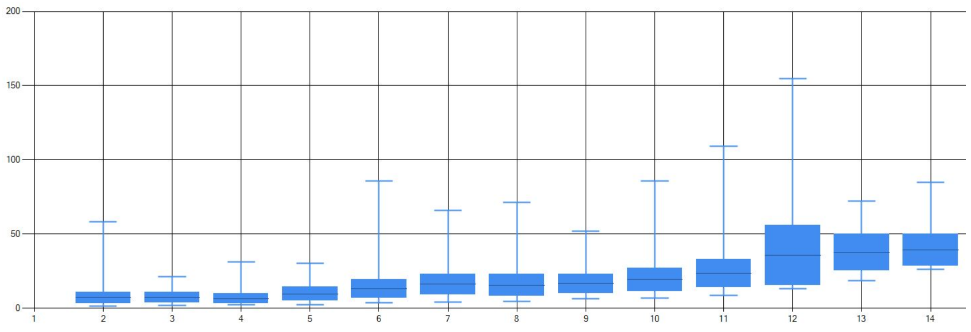

Figure 11: The average time to compute the optimal solutions (in ms), for different problem sizes (number of aircraft). The filled boxes show the mean ± std.dev, while the horizontal lines show the minimum and maximum observed run times.

7.2 Practical validation

For the purpose of validation exercise in the SESAR project, the above methods for continuous re-routing and re-scheduling, combined with guiding, were implemented in SINTEF's optimization software for ATM. This was integrated in a third party tower control system from Frequentis AG. The total integrated system was evaluated using Eurocontrol's airport simulator, using data based on real world traffic at Budapest Airport. The project validation report, as well as the ATM-specific characteristics of the test scenarios, can be found in the project data pack [24].

The controllers concluded that the approach of continuous re-scheduling combined with speed-controlling guidance can work well in practice, and would save communication workload. Also, they appreciated that the automatic suggestions of new routes to avoid deadlocks were useful, and released the controller from having to continuously look out for deadlock situations. However, it was noted that through this automation, it was easy for the controllers to lose some situational awareness, as they no longer needed to continuously keep every aircraft movement, and potential future conflicts, in their minds. However, the expected increase in airport traffic will increase the controller's workload significantly. We therefore believe that in the future, controllers must be supported by automated systems that detect, and offer solutions to, up-coming traffic conflicts. Our proposed system was a step in that direction.

8 Conclusions

The focus of this work was to demonstrate how airport ground control can be improved by a combination of dynamic (online) re-planning and automated guiding of aircraft. The practical evaluation by air traffic controllers indicated that the concept is promising, and that it would reduce the controllers' workload and thus enable higher airport efficiency even as traffic volumes are expected to rise. A note was made, however, that it is challenging to retain situational awareness when decision- and communication processes are partially automated.

In order to generate optimal plans we developed a innovative approach which exploits the concept of conflict region to model and handle conflicts caused by spacial proximity. This allowed us to define a disjunctive formulation for the scheduling problem, which we then solved by branch & bound. The overall solution algorithm was able to tackle real-life instances from Budapest airport, returning plans every few seconds, as required by the practical setting.

Regarding the concept of conflict diagrams, we have presented both the theoretical aspects necessary to develop our solution approach as well as a practical algorithm for computing conflict diagrams from general route and vehicle geometries (see the Appendix, Section A).

Future developments may go in several directions.

- Conflict diagrams can be generalized to three or even more vehicles.
- Control points could also model push-back operations.
- We only experimented with a medium-sized airport (Budapest) and our approach may not always scale well for very large airports. In this case, we expect that the efficiency of the approach can be substantially improved by exploiting various techniques from combinatorial optimization and integer programming (e.g. decomposition, reformulation, strong cuts, etc.). Then, it may be necessary to resort to a state-of-the-art MILP solver (such as GUROBI or CPLEX).
- For airports which do not make use of pre-defined taxi-routes, a sophisticated route generation procedure may be necessary.
- Airports that are very different from the one we studied may require modifications to our approach. For example, for an airport with areas without a well defined movement network, the possibility of guiding an aircraft along a certain path is more limited. For such airports, the presented approach must be extended to include rapid re-routing that takes actual aircraft movements into account.

- The inherent stochastic nature of the problem raises significant future research challenges.
- Future ground controller software must exploit work-reducing computational support while maintaining sufficient situational awareness for the controllers. How to achieve this is an interesting (cross-disciplinary) research question in itself.

Acknowledgments

The work presented in this paper was done in project "PJ03a SUMO" ([27]), and was partially funded through the EUROPEAN UNION's Single European Sky Joint Undertaking (SJU) as a part of the SESAR Programme [28]. The paper was also partially funded by the Research Council of Norway, through the project OPSTRA (p.nr.267554). Opinions expressed in this paper reflect the authors' findings only, and the SJU shall not be considered liable for them or for any use that may be made of the information contained herein. The authors are grateful for the fruitful collaboration within the project consortium. In particular, we wish to thank the participating air traffic controllers for their invaluable input, HungaroControl for providing data for Budapest airport, and, especially, Alina Graf and the team at Frequentis.

References

[1] Ravindra K Ahuja, Thomas L Magnanti, James B Orlin, and K Weihe. *Network flows: theory, algorithms and applications.* Prentice Hall, 1993.

[2] Florent Altché, Xiangjun Qian, and Arnaud de La Fortelle. Time-optimal coordination of mobile robots along specified paths. *CoRR*, abs/1603.04610, 2016.

[3] Florent Altché, Xiangjun Qian, and Arnaud de La Fortelle. An algorithm for supervised driving of cooperative semi-autonomous vehicles (extended). *CoRR*, abs/1706.08046, 2017.

[4] Jason AD Atkin, Edmund K Burke, and Stefan Ravizza. The airport ground movement problem: Past and current research and future directions. In *Proceedings of the 4th International Conference on Research in Air Transportation (ICRAT), Budapest, Hungary*, pages 131–138, 2010.

[5] Sandeep Badrinath, Hamsa Balakrishnan, Emily Joback, and Tom G Reynolds. Impact of off-block time uncertainty on the control of airport surface operations. *Transportation Science*, 54(4):920–943, 2020.

[6] Sandeep Badrinath, Hamsa Balakrishnan, Ji Ma, and Daniel Delahaye. Comparative analysis of departure metering at united states and european airports. *Journal of Air Transportation*, 28(3):93–104, 2020.

[7] Hamsa Balakrishnan. Control and optimization algorithms for air transportation systems. *Annual Reviews in Control*, 41:39–46, 2016.

[8] Egon Balas. Disjunctive programming. In *Annals of Discrete Mathematics*, volume 5, pages 3–51. Elsevier, 1979.

[9] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European journal of operational research*, 112(1):3–41, 1999.

[10] Sonia Cafieri and David Rey. Maximizing the number of conflict-free aircraft using mixed-integer nonlinear programming. *Computers & Operations Research*, 80:147–158, 2017.

[11] European Commission. Single european sky atm research joint undertaking. https://www.sesarju.eu.

[12] Julien Guépet, Olivier Briant, Jean-Philippe Gayon, and Rodrigo Acuna-Agost. The aircraft ground routing problem: Analysis of industry punctuality indicators in a sustainable perspective. *European Journal of Operational Research*, 248(3):827–839, 2016.

[13] Julien Guépet, Olivier Briant, Jean-Philippe Gayon, and Rodrigo Acuna-Agost. Integration of aircraft ground movements and runway operations. *Transportation research part E: logistics and transportation review*, 104:131–149, 2017.

[14] Harshad Khadilkar and Hamsa Balakrishnan. Network congestion control of airport surface operations. *Journal of Guidance, Control, and Dynamics*, 37(3):933–940, 2014.

[15] Dag Kjenstad, O. Kloster, K.F. Pettersen, M. Smedsrud, C. Schulz, P. Schittekat, and T.E. Nordlander. Simulation of rail replacement bus service in oslo. In *Lecture Notes in Management Science*, volume 8, pages 16–21. ORLAB, 2016.

[16] Dag Kjenstad, Carlo Mannino, Patrick Schittekat, and Morten Smedsrud. Integrated surface and departure management at airports by optimization. In *Modeling, Simulation and Applied Optimization (ICMSAO), 2013 5th International Conference on*, pages 1–5. IEEE, 2013.

[17] Elisabeth Lübbecke, Marco E Lübbecke, and Rolf H Möhring. Ship traffic optimization for the kiel canal. *Operations Research*, 2019.

[18] Carlo Mannino and Alessandro Mascis. Optimal real-time traffic control in metro stations. *Operations Research*, 57(4):1026–1039, 2009.

[19] Carlo Mannino, Andreas Nakkerud, and Giorgio Sartor. Air traffic flow management with layered workload constraints. *Computers & Operations Research*, 127:105159, 2021.

[20] Alessandro Mascis and Dario Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002.

[21] Mayara Condé Rocha Murça. A robust optimization approach for airport departure metering under uncertain taxi-out time predictions. *Aerospace Science and Technology*, 68:269–277, 2017.

[22] A Nakahara, TG Reynolds, T White, C Maccarone, and R Dunsky. Analysis of a surface congestion management technique at new york jfk airport. 11th american institute of aeronautics and astronautics aviation technology. In *Integration, and Operations Conference, Virginia Beach, VA*, pages 20–22, 2011.

[23] Manfred Padberg. *Linear optimization and extensions*, volume 12. Springer Science & Business Media, 2013.

[24] PJ03a - 01 consortium. Solution pj.03a-03: V2 data pack, 2019. "https://ec.europa.eu/research/participants/documents/", Last accessed on 2019-12-31.

[25] Marcella Samà, Andrea D'Ariano, Paolo D'Ariano, and Dario Pacciarelli. Air traffic optimization models for aircraft delay and travel time minimization in terminal control areas. *Public Transport*, 7(3):321–337, 2015.

[26] Ioannis Simaiakis, Harshad Khadilkar, Hamsa Balakrishnan, Tom G Reynolds, and R John Hansman. Demonstration of reduced airport congestion through pushback rate control. *Transportation Research Part A: Policy and Practice*, 66:251–267, 2014.

[27] SESAR Joint Undertaking. Pj.03 integrated surface management. http://https:www.sesarju.eu/projects/sumo. "Last visited 17.04.2020".

[28] SESAR Joint Undertaking. *SESAR Solutions Catalogue 2019*. Publication Office of the European Union, 2019.

[29] Mario Vanhoucke, Erik Demeulemeester, and Willy Herroelen. An exact procedure for the resource-constrained weighted earliness–tardiness project scheduling problem. *Annals of Operations Research*, 102(1-4):179–196, 2001.

[30] Adan E Vela, Erwan Salaun, Senay Solak, Eric Feron, William Singhose, and J-P Clarke. A two-stage stochastic optimization model for air traffic conflict resolution under wind uncertainty. In *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*, pages 2–E. IEEE, 2009.

Appendices

A Constructing conflict regions

In this section, we describe an algorithm for constructing conflict diagrams for practical applications. Consider a pair of vehicles a and b with trajectories R^a and R^b. We assume that R^a and R^b are given as polyline paths, and the shapes and dimensions of a and b are given as polygons. Furthermore, we assume that a rule is given for computing the position and orientation of a's shape polygon for any position $x^a \in [0, L^a]$. For example, a simple rule could specify that the shape polygon be placed with its center on $R^a(x^a)$ (the point at distance x^a along R^a), oriented so that the shape's tail points toward $R^a(x^a - l)$, for some fixed distance l. (The definition of $R^a(x^a)$ is extended to negative x^a along the line that the first segment of R^a is part of.) More advanced rules can be given that model the vehicle movement in a physically more realistic way. For any such rule, we require that the point $R^a(x^a)$ is contained in the polygon computed for position x^a.

So, we have a function $S^a : [0, L^a] \to$ (polygons in $\mathbb{R}^2$), which maps a position x^a to the polygon occupied by a, and similarly, S^b maps x^b to b's polygon. Define the function $d : B \to \mathbb{R}$ as follows: If the polygons $S^a(x^a)$ and $S^b(x^b)$ overlap, then $d(x^a, x^b) = -\sqrt{A}$, where A is the area of overlap, and we describe this situation as a *conflict*. Otherwise, $d(x^a, x^b)$ is the distance between the closest points in $S^a(x^a)$ and $S^b(x^b)$. The function d is straightforward to compute, and we have that $(x^a, x^b) \in C$ iff $d(x^a, x^b) < 0$. If the application requires a safety margin, this rule would be amended to say that $(x^a, x^b) \in C$ iff $d(x^a, x^b) < d_{min}$, for some threshold d_{min}.

With these preliminaries, we can describe the algorithm for constructing a convex polygonal approximation to C. The major tool used in the construction is sampling the function d. Besides indicating whether a point is in C or not, d gives an indication of the distance to the boundary of C, which improves the behaviour of the algorithm. Note that while the algorithm creates good approximations in practice, it does not provide specific guarantees. The steps in the algorithm are:

1. Identify conflict seeds
2. Construct initial polygons
3. Refine the polygons

1. Identifying conflict seeds. The first step is, for each conflict, to identify some point in the conflict region. We find these points, which are called *conflict seeds*, in two ways.

The first way is to look for actual intersections between R^a and R^b. If $R^a(x^a) = R^b(x^b)$ for some $x^a \in [0, L^a], x^b \in [0, L^b]$, then $S^a(x^a)$ and $S^b(x^b)$ necessarily overlap, so we can add any point (x^a, x^b) where this holds as a conflict seed. Computing the intersection points between two piecewise linear curves is straightforward.

The second way is to look for points where R^a and R^b come close but do not actually intersect. To this end, we find the points $(x^a, x^b) \in B$ where the distance between $R^a(x^a)$ and $R^b(x^b)$ has a local minimum whose value is positive, but smaller than the sum of the shape diameters of a and b. This is also a straightforward computation for polylines. A point (x^a, x^b) found in this way is not necessarily part of a conflict, so we add it as a conflict seed only if $d(x^a, x^b) < 0$.

2. Constructing initial polygons For each conflict seed, we construct an initial polygon by finding four points on the boundary of the conflict region C, one in each cardinal direction.

Observe that for any point p on the boundary of C, either $d(p) = 0$, or p is on the boundary of B. This is the basis for the following algorithm for finding a point on the conflict region boundary, which is used both in this step and in step 3 below. The algorithm samples d along a line, at successive points with offset o, until either the line leaves the domain B, or d changes sign. In the former case, the result is the intersection point between the line and the boundary of B. In the latter case, the result is a point on the conflict region boundary.

Let $p \in B$ be a point and o a vector offset:

Algorithm A.1: FINDBOUNDARYPOINT(p, o)

```
$k \leftarrow 0$
**repeat**
 $k \leftarrow k + 1$
 **if** $(p + ko) \notin B$
   **then return** $(Intersection(p, p + ko, \delta B))$
**until** $d(p + ko) * d(p) \leq 0$
$\alpha \leftarrow ZeroArgument(d(p + (k - \alpha)o), \alpha \in [0, 1])$
**return** $(p + (k - \alpha)o)$
```

In the above, $Intersection(p_1, p_2, P)$ finds the intersection point between the line segment from p_1 to p_2 and the (closed) polyline P, while $ZeroArgument(f(\alpha), \alpha \in [a, b])$ computes a zero of f in the interval $[a, b]$ numerically using the secant method.

The four points of the initial polygon are found by the above algorithm, taking p as the conflict seed and o as $(1, 0)$, $(0, 1)$, $(-1, 0)$ and $(0, -1)$.

3. Refining the polygon

The initial approximation to C is a convex polygon, where each vertex lies on the boundary of C. We now create a succession of better approximations to C, while preserving these properties. The following operations are performed in each iteration:

- Examine each edge of the polygon and determine any new vertices related to that edge.
- If no new vertices were found, terminate.
- Take the union of the existing polygon's vertices and the new vertices. The convex hull of this vertex set is the new approximation polygon.

The determination of new vertices related to an edge proceeds as follows:

1. Let p and q be the ends of the edge (thus also vertices of the polygon), in counter-clockwise order around the polygon, and let c be its midpoint.

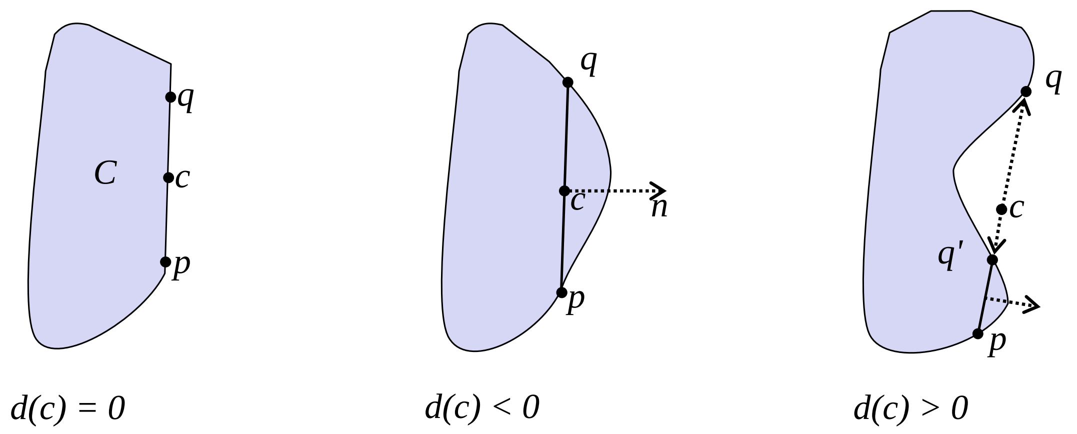

Figure 12: The cases considered in steps 3 to 5 of the refinement algorithm

2. If the distance between p and q is below a set tolerance, the polygon is considered detailed enough and no new vertex is created.
3. Evaluate $d(c)$. If $d(c) = 0$, c is on the boundary of C. The edge is considered straight, and no new vertex is created.
4. If $d(c) < 0$, c is inside C. A new vertex is created at $\mathit{FindBoundaryPoint}(c, n)$, where n is the right unit normal to the line from p to q.
5. If $d(c) > 0$, c is outside C, but there might be other points along the edge that are inside. To check for this, we compute $q' = \mathit{FindBoundaryPoint}(c, u)$, where u is a unit vector in the direction from c to p. If $q' \neq p$, we add any new vertices that result from considering (p, q') an edge of the polygon. Similarly, we find $p' = \mathit{FindBoundaryPoint}(c, -u)$ and if $p' \neq q$, add any new vertices resulting from the edge (p', q).

B Extending conflict regions

We show how to extend a conflict region to include points which are not strictly conflict points but still cannot belong to any feasible concurrent trajectory. Informally, these are points from where it would not be possible to reach the destination without intercepting the conflict region or points which are only reachable going through the conflict region.

By adding all such points to the conflict region we can define what we will call an *extended conflict region*.

To this end, we consider again the concurrent space B associated with the concurrent positions of two vehicles a and b, and let $q \in B$ (Fig. 13). Point $q = (q^a, q^b)$ identifies four regions of B: *South-West* quadrant $SW(q) = \{(x^a, x^b) \in \mathbb{R}^2_+ : 0 \leq x^a \leq q^a, 0 \leq x^b \leq q^b\}$, *North-West* quadrant $NW(q) = \{(x^a, x^b) \in \mathbb{R}^2_+ : 0 \leq x^a \leq q^a, q^b \leq x^b \leq L^b\}$, *North-East* quadrant $NE(q) = \{(x^a, x^b) \in \mathbb{R}^2_+ : q^a \leq x^a \leq L^a, q^b \leq x^b \leq L^b\}$, *South-East* quadrant $SE(q) = \{(x^a, x^b) \in \mathbb{R}^2_+ : q^a \leq x^a \leq L^a, 0 \leq x^b \leq q^b\}$.

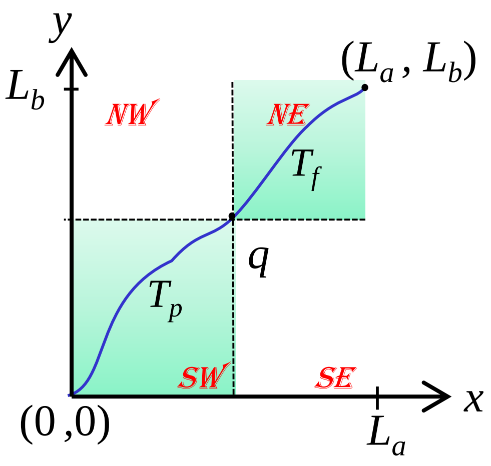

Figure 13: The four regions of B identified by a point q. Preceding and following trajectory through point $q \in T$.

Consider a feasible concurrent trajectory T going through q. By Property 2.1, each point in T belongs either to the *preceding trajectory* $T_p(q) = \{(x^a, x^b) \in T : x^a \leq q^a, x^b \leq q^b\}$ or to the *following trajectory* $T_f(q) = \{(x^a, x^b) \in T : x^a \geq q^a, x^b \geq q^b\}$. T_p and T_f share only point q. By definition, the preceding trajectory $T_p(q)$ is contained in $SW(q)$ whereas the following trajectory $T_f(q)$ is contained in $NE(q)$, as depicted in Fig. 13.

Lemma B.1 *Let $C \subset B$ be a conflict region, and let $q \in B \setminus C^\circ$ be a non-conflict point of B. Suppose there is no continuous line in $NE(q) \setminus C^\circ$ connecting q to (L^a, L^b). Then there is no feasible concurrent trajectory through q. Similarly, if there is no continuous line in $SW(q) \setminus C^\circ$ connecting $(0, 0)$ to q, then there is no feasible concurrent trajectory through q.*

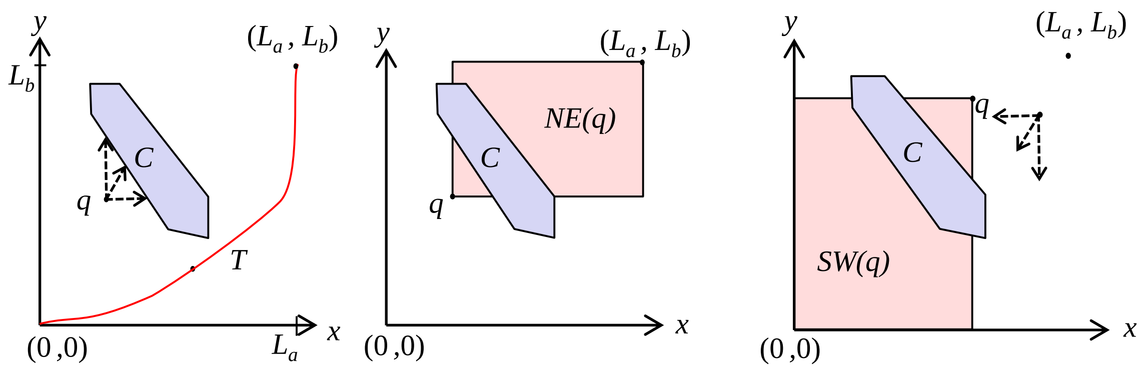

Figure 14: No feasible forward trajectory can pass through point q.

We define as the *extended conflict region* the union of all points satisfying the conditions of Lemma B.1 and the original conflict region C.