UiO : **University of Oslo**

Andreas Nakkerud

Integer Programming Approaches for Real-Time Traffic Management

Thesis submitted for the degree of Philosophiae Doctor

Department of Mathematics
Faculty of Mathematics and Natural Sciences

2022

*Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 2514*

ISSN 1501-7710

Cover: Hanne Baadsgaard Utigard.
Print production: Reprosentralen, University of Oslo.

To my wife and our family and friends

Preface

All the papers of this thesis have been written in collaboration with my supervisor Prof. Carlo Mannino of the University of Oslo and SINTEF. I have also collaborated closely with other researchers at SINTEF, most notably Dr. Giorgio Sartor, and with researchers at the University of Rome Tor Vergata, most notably Prof. Gianpaolo Oriolo and his then M.Sc. student Cristina Elena Filippi. They all have my deepest gratitude; I could not have done this without them.

> "Each of us is carving a stone, erecting a column,
> or cutting a piece of stained glass
> in the construction of something much bigger than ourselves."
>
> *Adrienne Clarkson*

First and foremost, I want to thank my supervisor, Professor Carlo Mannino. In one of our first meetings, Carlo told me that the next great idea could come from anyone, so I should never hesitate to speak my mind. Carlo truly lives by this belief, and he always takes the time to listen to my input. With this said, you can't be a good supervisor without occasionally rejecting bad ideas. Still, Carlo has the rare gift of making these occasions feel enlightening; it is impossible to leave a meeting with Carlo without being eager to get back to solving the problem.

I want to thank my friends and fellow students, Doctors Jørgen Trømborg, Anders Hafreager, Leif Harald Karlsen, Daniel Lupp, and Vidar Klungre, for proving that it can be done. Also, I would like to thank the members of the DataScience@UiO Innovation Cluster for providing a great community and forum for discussing our ideas and experiences. In particular, I would like to thank Professors Ingrid Kristine Glad and Arnoldo Frigessi for their interest and involvement in my work. On several pivotal occasions, they have helped shape the direction of my research. I would also like to extend special thanks to my fellow student Daniel Bakkelund. He has taught me much of both mathematics and programming in the course of our cross-disciplinary explorations.

I want to thank my teachers, Trond-Håkon Bjærke and Elbjørg Bjuving, for inspiring my love for mathematics and science. I also want to thank teacher, colleague, and friend Roger Antonsen, who has been an important mentor through most of my adult life.

I want to thank my brother, Erik Nakkerud, for many exciting and wild chess games. Especially during the pandemic, the occasional online chess match and accompanying phone call has been a much-appreciated relief.

I want to thank my parents, Anne K. Haugestad and Mads B. Nakkerud, for inspiring and supporting my interest in mathematics, science, and academic

pursuits since my childhood. They have always been great role models, showing me the value of knowledge and curiosity. Their example led me to see learning as among the most worthwhile of pursuits. I also want to thank my wife's parents for taking care of my wife and our daughter when I spent long hours working on this thesis.

Finally, I want to thank my wife, Elisabeth Nakkerud. She provides the bedrock of support on which I have built my life and work. We had a daughter in the spring of 2020, and we're expecting another child this winter; the joyful company of my family leaves me fulfilled in a way that makes it possible to get through anything.

Andreas Nakkerud
Oslo, January 2022

List of Papers

Paper I

C. Mannino, A. Nakkerud, G. Sartor, and P. Schittekat. ‘Hotspot Resolution with Sliding Window Capacity Constraints using the Path&Cycle Algorithm’. Published in *SESAR Innovation Days* **7** (2018).

Paper II

C. Mannino, A. Nakkerud, and G. Sartor. ‘Air Traffic Flow Management with Layered Workload Constraints’. Published in *Computers & Operations Research,* Volume 127 (2021).

Paper III

C. Mannino, A. Nakkerud ‘Optimal Train Rescheduling in Oslo Central Station’. Submitted to *Omega,* revision requested by editor.

Paper IV

A. Nakkerud ‘Rail Infrastructure Data for Oslo Central Station’. Submitted to *Data in Brief.*

Contents

Chapter 1

Introduction

"Nothing pertaining to humanity becomes us so well as mathematics."

Isaac Asimov

The topic of this thesis is the application of Mixed-Integer Linear Programming approaches to problems in real-time traffic management. With steady improvements in algorithm and solver design, the size and complexity of problems that can be solved by Integer Programming are ever-expanding. We focus on real-time scheduling problems in highly structured traffic systems, specifically aviation and railways.

Thesis: Integer programming approaches to scheduling can be made flexible and fast enough for applications in real-time traffic management.

For a scheduling approach to be suitable for real-time applications, it must be able to handle real-life instances in a short time. In both air traffic management and train dispatching, solution times of a minute or less may be required if an approach is to make its way into practical use.

1.1 Mathematical Optimization

Before the invention of computers, mathematicians were not overly concerned with the complexity of formulas used in proofs and results. Once a problem was solved, it was solved. The arithmetical hierarchy, developed independently by Kleene and Mostowski in the mid-1940s, is among the first works on classifying the efficiency of a mathematical formula. With the invention of the Turing machine. In 1965, Alan Cobham published a paper titled *The intrinsic computation difficulty of functions*, where he suggested the modern definition of easy, or *tractable,* problems. During the late 20th century, complexity theory grew into a field of its own, and large classes of very difficult optimization problems were discovered.

Linear programming, a technique for solving large sets of linear inequalities in real variables, has been around in some form since Isaac Newton but was developed into the modern science we know today in the 1930s and 1940s. In the 1960s, algorithms for solving linear programs were extended to integer variables. With this, mathematical optimization could be used to attack some of the difficult problems that were cropping up in complexity theory.

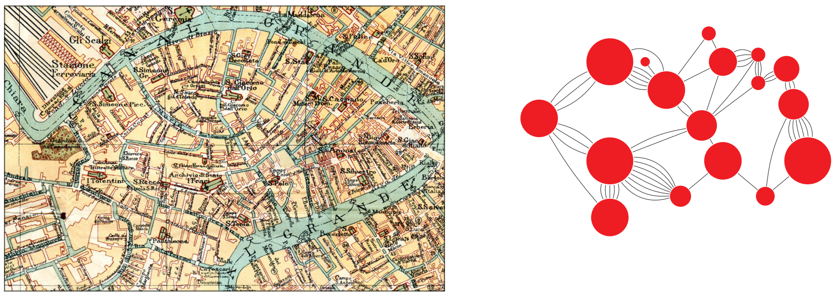


Figure 1.1: The city of Venice is famous for its bridges and canals. **Left:** Part of an old-school map of Venice. **Right:** A graph representation of the Venetian islands and bridges.

1.2 Easy Problems and Hard Problems

Imagine that Alice and Bob are going to Venice. They both love canals, but while Alice wants to see all the islands, Bob wants to see all the bridges. Of course, if Alice tags along with Bob, she will see all the islands; they all have bridges. Though, while Bob will enjoy walking endlessly around Venice, spending quality time on bridges, Alice may get tired of visiting the same islands over and over. Alice wants to know if it is possible to visit every island exactly once. Bob does not mind repeating islands but wants to visit every bridge exactly once. Figure 1.1 shows a map of Venice city center along with a graph representation of its islands and bridges.

For Bob, the problem turns out to be easy. He is trying to find what is known as an *eulerian path*. Alice is not as lucky; what she needs to find is a *hamiltonian path*. A simple inspection of each island can solve the first problem. For the second problem, no known general solution is known that is significantly better than checking each possible sequence of islands, the number of which is exponential in the number of islands.

At first glance, it is sometimes hard to tell which optimization problems are simple and which are difficult. Problems that appear easy may hide deep complexity, and problems that appear hard may contain structures that greatly simplify them. Of course, the problems of finding eulerian and hamiltonian paths are very familiar to mathematicians familiar with graph theory, but in practice, the difference is rarely as clear-cut.

1.2.1 Some Hard Problems can be Easy (Sometimes)

When crossing a bridge has a cost, we can look for the cheapest way to visit each island exactly once. This problem is famously known as the Travelling Salesperson Problem (TSP). We know TSP to be a hard problem because it can be used as a generalization for a large class of very hard problems. Formally, we say TSP is NP-hard since it generalizes any problem in the class NP.[1] It is widely believed that no problem in NP can have an efficient solution, which means TSP cannot have an efficient solution; an efficient solution for TSP would be an efficient solution for any problem in NP.

The formal complexity classification of a problem, such as NP or NP-hard, is fundamental from a theoretical perspective. In practice, these classifications give solid indications of how difficult the problem is to solve. However, problems with a high theoretical difficulty rating may still be solvable in practice. Many of the applications of Combinatorial Optimization is focused on solving NP-hard problems in practice.

In practice, the Travelling Salesperson Problem is usually given as a set of cities that should be visited, with travel costs given between any pair of cities. While TSP is NP-hard, this is only a statement about worst-case instances of the problem. It could be that practical instances were all straightforward to solve. This did not appear to be the case for TSP. In 1962, Proctor & Gamle offered a prize of \$10,000 for the optimal solution to a TSP instance of only 33 cities [Coo12]. This problem was already very difficult since no known approach was significantly faster than checking every ordering of the cities. For 33 cities, that is over 10^{35} orderings. For comparison, the age of the universe is about 10^{20} seconds.

Barring an exceptional breakthrough shattering our understanding of theoretical complexity, solving the TSP in general requires exponential computational resources; for every city added to the problem, the computational requirements double. If this is true in practice, even with Moore's law promising computing power growing exponentially more available, we can only expect to add one or two cities to our problems every year.

The Integer Programming community came to the rescue of the traveling salesperson. By 2006, the record for the largest solved TSP instance was an optimal tour of 85,900 cities. This record was set by the Concorde solver developed by William Cook et al. The number of orderings, in this case, is 10^{386521}, a number beyond imagining. It was not brute force but rather an understanding of some structure in the problem that brought the record to these unbelievable heights.

The travel distance between cities on a map is not randomly distributed. Instead, it is usually very closely correlated with the straight-line distance between the cities. This means that the shortest tour through a collection of cities likely will pass between neighboring cities more often than it jumps around.

What happens then if we take away this structure? What if we construct a map full of magical portals, strange barriers, and strict travel restrictions? If we allow arbitrary distances between pairs of cities, the TSP reverts to the problem of finding the shortest hamiltonian path in an arbitrary graph. In which case, we may be back to struggling with 33 cities.

[1]NP-complete is another common class of problems. This class contains the easiest NP-hard problems: the problems that generalize NP while also belonging to NP. That is, NP-complete is the intersection of NP and NP-hard. We use NP-hard here since it is a more general class, although many of the problems we discuss have versions that are NP-complete.

While the Concorde is an enormously impressive piece of software, it does not prove anything about the worst-case instances of the TSP. Sylvia Boyd and colleagues at the University of Ottawa discovered that the Petersen graph could form the basis for very tough instances of TSP. In one case, Concorde took over 11 hours to solve an instance with only 56 nodes, according to Boyd.

1.3 Combinatorial Optimization

The Travelling Salesperson Problem is one of many that falls under the purview of *combinatorial optimization*. Combinatorics is the study of orderings and combinations, and in combinatorial optimization, we study optimization problems where the space of possible solutions is built on combinatorial principles.

An illuminating example of combinatorial optimization is the so-called *knapsack problem*. In this problem, we are given the task of filling a knapsack for maximum value while respecting the weight limit of the knapsack. Each item we can choose from has a given weight and value.

In order to state the knapsack problem formally, we let $\mathcal{S}$ be the set off all the available items. For an item $s \in \mathcal{S}$, let C_s be the item's value, and W_s the weight. We let $\mathcal{W}$ be the maximum weight allowance of the knapsack and denote by S the set of items selected for the knapsack. The knapsack problem is as follows.

$$\begin{aligned} \textbf{maximize} \quad & \textstyle\sum_{s \in S} C_s \\ \textbf{subject to} \quad & \textstyle\sum_{s \in S} W_s \leq \mathcal{W} \\ & S \subseteq \mathcal{S} \end{aligned} \tag{1.1}$$

1.3.1 Integer Programming

Integer programming (IP) is one of the most powerful tools for solving combinatorial optimization problems. First, we must state our problem using integer variables. We let $\mathcal{S} = \{s_1, \dots, s_n\}$, and introduce the binary variables y_i for $1 \leq i \leq n$. We let $y_i = 1$ when $s_i \in S$, that is, when s_i is chosen to be in the knapsack.

$$\begin{aligned} \textbf{maximize} \quad & \textstyle\sum_{i=1}^{n} C_i y_i \\ \textbf{subject to} \quad & \textstyle\sum_{i=1}^{n} W_i y_i \leq \mathcal{W} \\ & y_i \in \{0, 1\} \qquad\qquad i \in \{1, \dots, n\} \end{aligned} \tag{1.2}$$

The above set of equations is an integer *linear* program (ILP) since both the objective is a linear function and all the constraints are linear. If we add

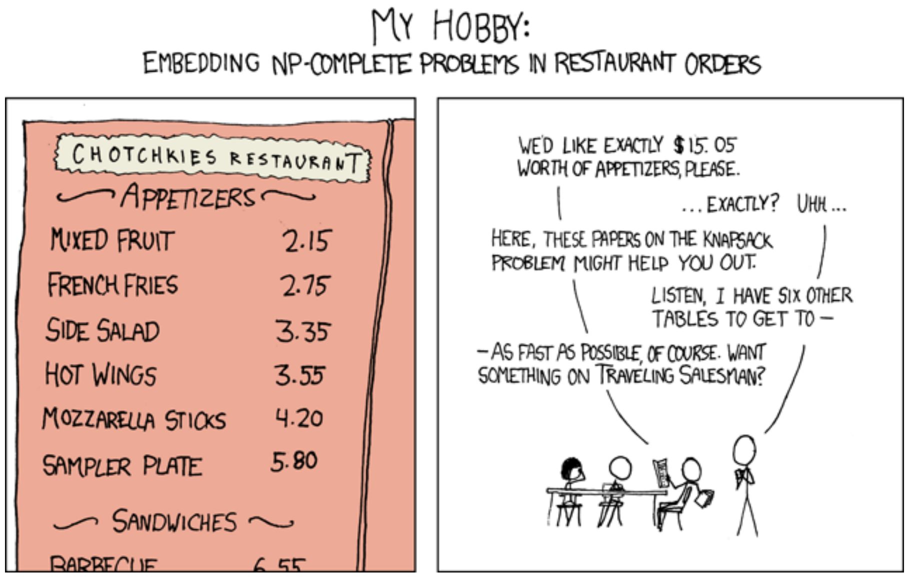

MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS
CHOTCHKIES RESTAURANT
APPETIZERS
MIXED FRUIT 2.15
FRENCH FRIES 2.75
SIDE SALAD 3.35
HOT WINGS 3.55
MOZZARELLA STICKS 4.20
SAMPLER PLATE 5.80
SANDWICHES
BARBECUE 6.55
WE'D LIKE EXACTLY $15.05 WORTH OF APPETIZERS, PLEASE.
...EXACTLY? UHH...
HERE, THESE PAPERS ON THE KNAPSACK PROBLEM MIGHT HELP YOU OUT.
LISTEN, I HAVE SIX OTHER TABLES TO GET TO —
— AS FAST AS POSSIBLE, OF COURSE. WANT SOMETHING ON TRAVELING SALESMAN?

Figure 1.2: XKCD #287 (NP-complete) by Randall Munroe (CC BY-NC 2.5).

continuous variables to the problem, we get a *mixed* integer linear program (MILP), which is the type of optimization model focused on in this thesis.

While the knapsack problem may not sound all too hard initially, it is, in fact, NP-hard. Figure 1.2 shows a comic by Randall Munroe highlighting how difficult even a simple instance of the knapsack problem can be. In the version of the knapsack problem in the comic, the task is not to maximize value but to find an exact match for the capacity of the knapsack. Also, items may be selected multiple times.

For the curious, the solution to the problem in Figure 1.2 is one mixed fruit, two hot wings, and one sampler plate. In an interview with the Mathematical Association of America, Munroe admitted missing a trivial solution to the problem: seven mixed fruit.

Integer programming (and mixed-integer linear programming) is a very powerful tool because many general-purpose solvers and specialized algorithms have been developed for solving IPs (and MILPs).

1.4 Scheduling Problems

One of the widespread applications of mathematical optimization is *scheduling problems*. In *job-shop scheduling*, for example, we are given a set of *jobs* or *work orders* to schedule. The schedule can be subject to several resource constraints, such as deadlines and personnel and equipment availability. The objective may be to minimize overtime, minimize deadline overruns, or maximize the number of completed work orders before a set time.

In the following example, we are given n jobs, and our task is to schedule as many as possible in a week of 5 days. Each job takes a whole day. E is the set of jobs requiring an engineer, and M is the set of jobs requiring a mechanic. $N_E(d)$ and $N_M(d)$ are, respectively, the number of engineers and mechanics available on day d. We define binary variables y_i^d, where $y_i^d = 1$ if we schedule job i on day d. The following is an IP *formulation* for a simple version of the job-shop scheduling problem.

$$\begin{array}{lll}
\textbf{maximize} & \sum_{i=1}^{n}\sum_{d=1}^{5} y_i^d & \\
\textbf{subject to} & \sum_{i\in E} y_i^d \le N_E(d) & d \in \{1,\dots,5\} \\
 & \sum_{i\in M} y_i^d \le N_M(d) & d \in \{1,\dots,5\} \\
 & \sum_{d=1}^{5} y_i^d \le 1 & i \in \{1,\dots,n\} \\
 & y_i^d \in \{0,1\} & i \in \{1,\dots,n\}, d \in \{1,\dots,5\}
\end{array} \tag{1.3}$$

Given a problem statement, there may be many possible formulations, each with advantages and disadvantages. The basis for any formulation is the choice of variables used to model decisions. These variables must be chosen so that we can express constraints and objectives as simply as possible. In the case of linear programming, the constraints and objectives must be linear functions in the variables.

We can extend the example (1.3) with additional types of constraints. For example, if job j must be completed before job k, we can add the constraints

$$\begin{aligned}
y_k^1 &= 0 \\
y_k^2 &\le y_j^1 \\
y_k^3 &\le y_j^1 + y_j^2 \\
y_k^4 &\le y_j^1 + y_j^2 + y_j^3 \\
y_k^5 &\le y_j^1 + y_j^2 + y_j^3 + y_j^4
\end{aligned} \tag{1.4}$$

That is, job k cannot be done on day 1. Furthermore, it can only be done on one of the other days if job j is completed on one of the preceding days.

A different formulation could help us simplify (1.4). If we let $z_i \in \{1,2,3,4,5\}$ be the day we schedule job i, then instead of (1.4), we write simply

$$z_k \ge z_j + 1 \tag{1.5}$$

The problem with this choice of variable is that the rest of the formulation becomes more complicated. We can no longer use the variables directly in a count, as we have done in (1.3).

Finding good IP (or MILP) formulations for scheduling problems is the major challenge in applying mathematical optimization to scheduling. A good

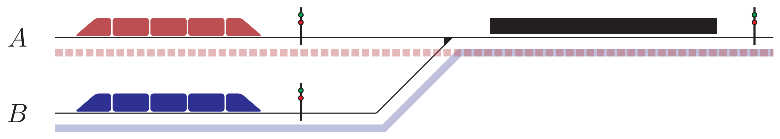


Figure 1.3: Two railway lines merging in a train station. Train dispatchers must decide the order of trains entering the station's one platform track.

formulation must permit modeling of all constraints and objectives and be suited to some solution approach—for example, the use of a general-purpose solver.

As modern solvers, like CPLEX and Gurobi, become better at exploiting structure in the formulations they solve, creating formulations with a clear structure becomes essential.

In (1.3), we handle scheduling conflicts by limiting the number of simultaneous jobs requiring the same personnel. In transport scheduling, we get similar constraints for trains in train stations, busses in bus terminals, and cargo trucks in loading bays.

1.4.1 Scheduling in Traffic Management

In the job-shop scheduling example in the previous section, we intentionally created a problem that was well suited to a *time-indexed* formulation. In these formulations, time is divided into discrete slots or windows. In the job-shop example, each slot is a full workday.

If we want to apply time-indexed formulations to traffic scheduling, we will need to use much shorter time slots. For trains in a railway station, time slots of 1 minute or shorter are often required. Scheduling 24 hours will then require 1440 time slots, compared to the five time slots used in (1.3). Even so, there are many scheduling applications where time-indexed formulations work well.

Figure 1.3 shows an example of a single platform train station with two incoming lines and one outgoing line. We let $\mathcal{A}$ be the set of trains to schedule and $\mathcal{H}$ be the planning horizon. That is, $\mathcal{H}$ is a set of time indexes, one for each time slot. If $a \in \mathcal{A}$ is a train and $t \in \mathcal{H}$ a time slot (or time index), then we let $x_{at} \in \{0, 1\}$ be a decision variable, where $x_{at} = 1$ if a arrives at the station platform in slot t. Furthermore, we let c_{at} be the cost of a arriving at t. Typically, this cost will be zero until the train becomes delayed. We let D_a be the dwelling time of train a. That is, the time a must wait at the platform before departing.

In order to create an IP formulation for this train scheduling problem, we introduce the set $T_{at} = \{(t - D_a + 1), \dots, t\}$. That is, T_{at} is the set of arrival times for a that would cause a to be at the platform at time t. If the train arrives at $t - D_a$, it would depart just before t. We get the following IP formulation.

$$\begin{array}{lll} \textbf{minimize} & \sum_{a\in\mathcal{A}}\sum_{t\in\mathcal{H}} c_{at}x_{at} & \\ \textbf{subject to} & \sum_{t\in\mathcal{H}} x_{at} = 1 & a\in\mathcal{A} \\ & \sum_{a\in\mathcal{A}}\sum_{t'\in T_{at}} x_{at'} \le 1 & t\in\mathcal{H} \\ & x_{at}\in\{0,1\} & a\in\mathcal{A}, t\in\mathcal{H} \end{array} \tag{1.6}$$

When we are scheduling only in a station, we must likely respect the order of the incoming trains on each line. We let $(a_1,\dots,a_n)$ be the sequence of trains coming in on line A, and $(b_1,\dots,b_m)$ be the sequence of trains coming in on line B. We can preserve the order of the trains on line A by adding the following constraints to (1.6):

$$\sum_{t\in\mathcal{H}}[t+D_{a_{i-1}}]x_{a_{i-1}t} \le \sum_{t'\in\mathcal{H}} t'x_{a_it'} \qquad i\in\{2,\dots,n\} \tag{1.7}$$

and similar for B. Since each train enters the station exactly once, the sum on the left-hand side has only one non-zero term. That is, $t+D_{a_{i-1}}$, where$t$ is the time train a_{i-1} enters the station and $D_{a_{i-1}}$ is the dwelling time of that train. The right-hand side also has a single non-zero term, which is t', where t' is the time a_i enters the station. Thus, the constraint ensures that each train must enter the station no sooner than the preceding train on the same line has left. Of course, any number of trains from the other line may come between the two trains.

We can further expand Equation (1.6) to cover additional line segments and stations. However, this would require the introduction of even more variables, along with additional constraint types. For example, we would likely need to add the option of a train remaining at the station for longer than the minimum dwelling time.

1.4.2 Continuous-Time Formulations

Different problems may call for different formulations. *Continuous-time* formulations are an alternative to time-indexed formulations for scheduling problems in traffic management. We revisit the station in Figure 1.3, and this time build a MILP formulation around the continuous-time variables t_a. For a train $a\in\mathcal{A}$, t_a is the time that train enters the station. If train a enters the station before train b, then we represent this by the constraint

$$t_b \ge t_a + D_a \tag{1.8}$$

In order to build a MILP formulation, we also need variables for deciding the order of trains. For trains $a,b\in\mathcal{A}$, we introduce the binary variables $y_{ab}, y_{ba}\in\{0,1\}$, where $y_{ab}=1$ is train a arrives in the station before train b. We combine these variables into the *disjunctive constraint*

$$\begin{array}{ll} t_b \ge t_a + D_a & \text{if } y_{ab}=1 \\ t_a \ge t_b + D_b & \text{if } y_{ba}=1 \end{array} \tag{1.9}$$

Since the two inequalities are equal to the order of the train names, we will often list only one in a formulation.

In order to use (1.9), we must first write it in a linear form. Taking the first of the two inequalities, we arrange it as follows:

$$t_b - t_a - D_a \geq 0 \qquad \text{if } y_{ab} = 1 \tag{1.10}$$

Notice that even when the order of the trains is wrong (or the trains overlap), the left-hand side of this inequality becomes negative. However, the magnitude of the left-hand side will never exceed the planning horizon of the model, at least not by much. After all, both trains must be scheduled to arrive within the planning horizon.

If we let M be some large number, maybe twice the length of the planning horizon, we can rewrite (1.10) as

$$t_b - t_a - D_a \geq -M(1 - y_{ab}) \tag{1.11}$$

If $y_{ab} = 0$, then the right-hand side of this inequality becomes $-M$. This is such a negative number that the left-hand side will always be larger, and the constraint is effectively inactivated. If $y_{ab} = 1$, on the other hand, the right-hand side becomes 0, and the constraint is active.

The trick shown in (1.11) is known as the big-M trick, and formulations using this trick are known as big-M formulations. The trick is very powerful, but it has its drawbacks. Adding very large coefficients to a MILP formulation can slow down the solution process, throw off heuristics that cut off irrelevant parts of the solution space, and even cause numerical instability in the solver.

As before we let $(a_1, \ldots, a_n)$ be the sequence of incoming trains on line A, and $(b_1, \ldots, b_m)$ the same for line B. We let $\mathcal{C}$ be the set of pairs of trains on different lines. That is,

$$\mathcal{C} = \{(a_i, b_k) \mid i \in 1 \ldots n, k \in 1 \ldots m\} \tag{1.12}$$

For each train $a \in \mathcal{A}$, we let S_a be the scheduled arrival time of a. We can then write the following big-M-based MILP formulation for our train scheduling example:

$$\begin{array}{rll}
\textbf{minimize} & \sum_{a \in \mathcal{A}} \eta_a & \\
\textbf{subject to} & t_{a_i} \geq t_{a_{i-1}} + D_{a_{i-1}} & i \in 2 \ldots n \\
& t_{b_k} \geq t_{b_{k-1}} + D_{b_{k-1}} & k \in 2 \ldots m \\
& & \\
& t_b - t_a - D_a \geq -M(1 - y_{ab}) & (a, b) \in \mathcal{C} \\
& t_a - t_b - D_b \geq -M(1 - y_{ba}) & (a, b) \in \mathcal{C} \\
& & \\
& \eta_a \geq t_a - S_a & a \in \mathcal{A} \\
& & \\
& t_a \geq S_a & a \in \mathcal{A} \\
& y_{ab} \in \{0, 1\} & a, b \in \mathcal{A}
\end{array} \tag{1.13}$$

In the last lines, we define the variables used in the formulation. For continuous variables with constant lower or upper bounds, it is customary to state these bounds instead of the variable type, as we have done here for t_a. If a continuous variable r is unbounded, we write $r \in \mathbb{R}$.

With the continuous-time variables, it is very easy to extend this formulation to allow trains to wait at the station beyond their dwelling time. Instead of having a single continuous variable per train $a \in \mathcal{A}$, we introduce the arrival time t_a^A and departure time t_a^D variables. Since the arrival time determines the delay, we can use it in the objective. Of course, with minimum dwelling time, we must have

$$t_a^D \geq t_a^A + D_a \tag{1.14}$$

so that the train does not depart too early.

Now, if train b must wait for train a, we have

$$t_b^A \geq t_a^D \tag{1.15}$$

The arrival of b must be after the departure of a.

The advantage of this generalization is that we can now let the model respect constraint on the outbound line. If there were different outbound lines, the formulation then allows the prioritization of trains that can depart quickly after arrival. It is possible to model these same constraints by extending the time-indexed formulation (1.6). However, the extension is more natural and easy to understand with the continuous-time variables.

1.4.3 Scheduling in Highly Structured Traffic Systems

The previous section gave examples of MILP formulations for two different scheduling problems. Considering how different these formulations were, it may seem like different problems always call for different formulations. After all, train, car, airplane, and boat traffic behave very differently.

However, it turns out that several highly structured traffic systems behave similarly to railways. A high-speed railway connecting two cities shares many important constraints with a flight connection between the same cities. In both cases, the planning stage may be able to ignore much of what happens outside the stations or airports. In both cases, the capacity for waiting vehicles is limited, and a very limited number of vehicles can enter or leave simultaneously. Boat traffic in shipping canals may show similar constraints, while car traffic is likely to be too flexible and therefore too different.

When the scheduling of a traffic system is focused around some heavily restricted chokepoints, we can likely use scheduling approaches developed for railways. This thesis studies how a class of MILP formulations can be applied to congestion reduction in centralized air traffic control and real-time train dispatching in large passenger stations.

1.5 Decomposition Approaches

Big-M MILP formulations may be suited to various decomposition approaches. In this thesis, we study the effect of using an improved Bender's decomposition approach, called the Path&Cycle approach [LM19].

One of the drawbacks of the Path&Cycle approach is that it leads to a non-compact formulation. The approach eliminates big-M constraints but does so at the cost of replacing them with a potentially exponential number of new constraints.

In our experience, the decomposition approach was less effective for the combined in-station routing and dispatching problem. In Paper I, we compared a decomposition approach with a standard approach to scheduling problems in air traffic control. Having found the decomposition approach promising, we made a more extensive study of it in Paper II. In Paper III, we used a big-M approach without decomposition, as the non-compactness introduced by the decomposition became a problem.

1.5.1 Delayed Row Generation

In MILP formulations for train scheduling, a large number of constraints are needed to handle conflicts between trains, as can be seen in (1.13). The number of these disjunctive constraints is quadratic in the number of trains, whereas the number of the constraints is linear.

If the existing schedule results in only a few conflicts, we may get away with leaving out a large number of the disjunctive constraints. One approach, called *delayed row generation*, is to drop most or all the disjunctive constraints from the beginning the add them one by one whenever the current solution violates one. This process requires solving many intermediate optimization problems, but they are significantly smaller than the full problem. The delayed row generation problem works well when this process generates only a tiny fraction of the rows in the full formulation.

It is delayed row generation that also makes the Path&Cycle approach possible. While it may introduce an exponential number of constraints into the formulation, most of these may be superfluous in many cases. Again, we may start by dropping all of them and introducing them as needed.

Delayed row generation relies on the ability to separate violated constraints efficiently. That is, we need a fast way of checking if a given solution is, in fact, feasible or if there are any constraints it violates. Efficient constraint separation is possible for scheduling conflicts between trains or airplanes and for the constraints introduced by the Path&Cycle approach.

References

[Coo12] Cook, W. J. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation.* Princeton University Press, 2012.

[LM19] Lamorgese, L. and Mannino, C. “A Noncompact Formulation for Job-Shop Scheduling Problems in Traffic Management”. In: *Operations Research* vol. 67, no. 6 (2019), pp. 1586–1609.

Chapter 2

Theoretical Background

"Without mathematics, there's nothing you can do.
Everything around you is mathematics.
Everything around you is numbers."

Shakuntala Devi

In mathematical optimization, we study problems of the form

$$\min_{x \in X} f(x) \tag{2.1}$$

The different branches of mathematical optimization focus on various restrictions on X and f. In linear optimization, one of the most restrictive classes of mathematical optimization, both X and f must allow linear representation. Examples of more general classes of mathematical optimization are quadratic optimization and convex optimization. These other classes allow more expressive mathematical models at the cost of more demanding solution algorithms.

This chapter aims to outline the concepts that are central to the work presented in this thesis. Thus, many details are omitted when they do not directly bear upon the topics explored in this thesis. For a thorough introduction to mathematical optimization, see [BT97; WN99].

2.1 Linear Programming

Linear programming is a common approach to optimization problems that allow linear representation. It is a widely applied approaches to mathematical optimization. The normalized form of a *linear program* is

$$\text{(LP)} \qquad z = \min\{\mathbf{c}^T\mathbf{x} : A\mathbf{x} \geq \mathbf{b}, \mathbf{x} \in \mathbb{R}^n_+\}. \tag{2.2}$$

Linear programs can be solved in polynomial time using the ellipsoid method, proving they are computationally tractable. However, most linear problems are solved more efficiently using the simplex algorithm by George Dantzig, even though this algorithm may require exponential time.

Within linear programming, we often allow integrality constraints as a special type of non-linear constraint. If we represent a problem using only integer variables and linear expressions, the result is an *integer (linear) program*

$$\text{(IP)} \qquad z = \min\{\mathbf{c}^T\mathbf{x} : A\mathbf{x} \geq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^n_+\}. \tag{2.3}$$

If we represent a problem using a mix of continuous and integer variables, the result is a *mixed-integer (linear) program*

$$\text{(MIP)} \qquad z = \min\{\mathbf{c}^T\mathbf{x} + \mathbf{f}^T\mathbf{y} : A\mathbf{x} + B\mathbf{y} \geq \mathbf{b}, \mathbf{x} \in \mathbb{R}^n_+, \mathbf{y} \in \mathbb{Z}^p_+\}. \tag{2.4}$$

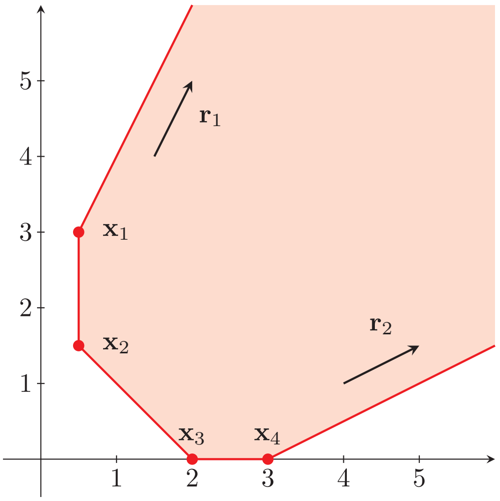


Figure 2.1: Unbounded polytope in $\mathbb{R}^2_+$ with 4 vertices and 2 extreme rays.

2.1.1 Simplex Algorithm

For the (continous) linear program (2.2), the *feasible region* is the set

$$P = \left\{ \mathbf{x} \in \mathbb{R}^n_+ : A\mathbf{x} \geq \mathbf{b} \right\}. \tag{2.5}$$

Such a subset of $\mathbb{R}^n_+$ bounded by the hyperplanes $A_i\mathbf{x} = b_i$, is called a *convex polytope* or simply *polytope*. Figure 2.2 shows a bounded 3-dimensional convex polytope.

When a convex polytope is bounded, it can also be defined as the convex hull of a set of critical points or vertices. A point x is a vertex of P if it cannot be written as a convex combination of two other points in P. That is, x is not a non-extreme point on any line in P. If X is the set of vertices of P, we write

$$P = \operatorname{conv}(X) = \left\{ \sum_{\mathbf{x}\in X} \lambda_{\mathbf{x}}\mathbf{x} : \sum_{\mathbf{x}\in X} \lambda_{\mathbf{x}} = 1, \lambda \in \mathbb{R}^X_+ \right\} \tag{2.6}$$

When a convex polytope in $P \subseteq \mathbb{R}^n_+$ is unbounded, there will be rays $\mathbf{r} \in \mathbb{R}^n_+$ such that for any $\mathbf{x} \in P$, we have $\mathbf{x} + \mu\mathbf{r} \in P$ for any $\mu \geq 0$. A ray in P is an *extreme ray* if it cannot be written as a convex combination of other rays in P. Figure 2.1 shows an example of an unbounded 2-dimensional polytope with 4 vertices and 2 extreme rays.

If a polytope P has vertices X and extreme rays R, we can write

$$P = \left\{ \mathbf{x} + \sum_{\mathbf{r}\in R} \mu_{\mathbf{r}}\mathbf{r} : \mathbf{x} \in \operatorname{conv}(X), \mu \in \mathbb{R}^R_+ \right\} \tag{2.7}$$

In linear programming, we typically define polytopes by bounding hyperplanes. That is, by sets of linear inequalities. While it is very natural to state optimization

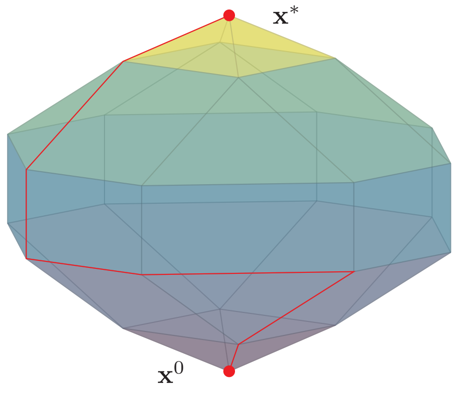


Figure 2.2: The simplex algorithm steps from vertex to vertex, going from some initial solution $\mathbf{x}^0$ to an optimal solution $\mathbf{x}^*$. Each step must be an improvement of the solution.

problems in terms of inequalities, it is natural to discuss the solution of these problems in terms of vertices and rays, as indicated by the following proposition.

Lemma: Let $P \subseteq \mathbb{R}^n$ be a convex polytope with vertices X and extreme rays R, let $\mathbf{c} \in \mathbb{R}^n$ such that $\mathbf{c}^T\mathbf{r} < 0$ for all rays $\mathbf{r} \in R$, and let $\mathbf{x}' \in X$. Then either

$$\mathbf{c}^T\mathbf{x}' = \min_{\mathbf{x}\in P} \mathbf{c}^T\mathbf{x} \tag{2.8}$$

or there is an edge in P connecting $\mathbf{x}'$ to $\mathbf{x}''$ where $\mathbf{c}^T\mathbf{x}'' > \mathbf{c}^T\mathbf{x}'$.

The simplex lemma guarantees that we can find an optimal solution to an LP by moving from vertex to vertex in the corresponding polytope, given that the LP is not unbounded. Figure 2.2 shows a sequence of vertices that could be produced by the simplex algorithm.

While the simplex algorithm is guaranteed to find an optimal solution to a bounded LP, it may require a vast number of steps. We can try to mitigate this by looking for the option giving the largest immediate improvement. However, there is no guarantee that such a greedy algorithm will find the shortest path. Although the worst-case performance of the simplex algorithm is poor, the algorithm turns out to be enormously efficient in practice.

To understand why we can use the n-dimensional hypercube as an illustration. It is bounded by the $2n$ hyperplanes $x_i \geq 0$ and $x_i \leq 1$ for $1 \leq i \leq n$, but has the 2^n vertices $\{0,1\}^n$. This feasible region is huge, but the path taken by simplex can have at most n steps; each step taken must improve the objective, which is to minimize $\mathbf{c}^T\mathbf{x}$.

The n-dimensional hypercube polytope has edges connecting pairs of vertices that differ in exactly one coordinate x_i. The simplex algorithm can follow ad

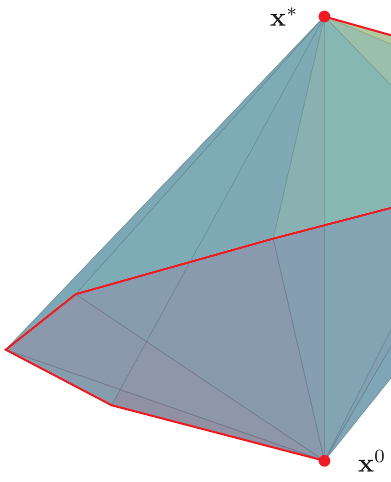


Figure 2.3: It is theoretically possible for simplex to find very long paths to the optimal solution, even when each step must improve the solution.

edge changing x_i from 1 to 0 if $c_i > 0$, and from 0 to 1 if $c_i < 0$. Thus, once variable x_i has been changed, it can never change back since this would worsen the objective. As a result, the simplex algorithm can take at most n steps.

Of course, the n-dimensional hypercube is an artificially simple case. The optimal solution could be determined directly by inspection of the objective as

$$x_i = \begin{cases} 1 & \text{if } c_i \leq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.9}$$

There will generally be edges between vertices that differ in several variables. Figure 2.3 shows a slightly more complex example where a terrible solution path is possible. The figure shows the convex hull of points on a spiral

$$P = \operatorname{conv}\left\{(\cos 2\pi z, \sin 2\pi z, z) : z \in \{i/8 : i = 0 \ldots 8\}\right\} \tag{2.10}$$

If we let $\mathbf{c} = (0, 0, \pm 1)$, then each vertex of P has an edge leading to the optimal vertex. Still, it is possible for the simplex algorithm to find a path visiting every vertex. This path is shown in Figure 2.3.

While the simplex algorithm is easy to conceptualize, it is not trivial to implement. However, many high-quality solvers are available commercially and as community projects. In our work, we rely entirely on solvers for solving LPs. For more details on the simplex algorithm, see [BT97].

2.1.2 Linear Programming Duality

A very important result in linear optimization is the strong duality theorem for linear programming. Given a linear program

$$z = \min\left\{\mathbf{c}^T\mathbf{x} : A\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\right\} \tag{2.11}$$

there exists a dual linear program

$$v = \max\left\{\mathbf{u}^T\mathbf{b} : \mathbf{u}^T A \leq \mathbf{c}^T, \mathbf{u} \geq \mathbf{0}\right\} \tag{2.12}$$

The strong duality theorem states that if either problem is feasible, then both are feasible and $z = v$.

Linear programming duality is useful for rewriting problems to an equivalent form. The weak duality theorem states that for any pair of feasible solutions $\mathbf{x}_i, \mathbf{u}_j$, we have

$$\mathbf{c}^T\mathbf{x}_i \geq \mathbf{u}_j^T\mathbf{b} \tag{2.13}$$

so that we can use dual linear programs to find bounds on the optimal solution value.

2.2 Mixed-Integer Programming

Integer Linear Programming is a branch of linear programming where we slightly relax the linearity requirement. We do this in a very controlled way by allowing integrality constraints on the variables of the linear program.

The resulting feasible region is no longer a polytope but instead a union of polytopes.

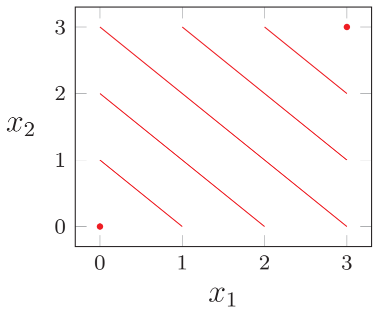
$$\begin{aligned} \textbf{minimize} \quad & c_1x_1 + c_2x_2 + fy \\ \textbf{subject to} \quad & x_1 + x_2 - y = 0 \\ & x_1, x_2 \in [0, 3] \\ & y \in \mathbb{Z}_+ \end{aligned} \tag{2.14}$$

We note that if we let $\mathbf{x} = (x_1, x_2, y)^T$, then the feasible region of (2.14) can be expressed as

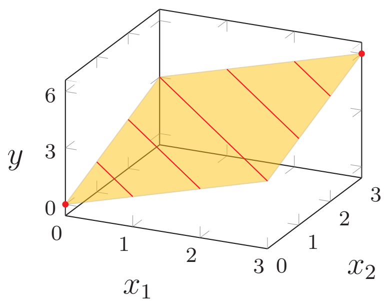
$$X = \{\mathbf{x} \in \mathbb{R}^3_+ : A\mathbf{x} \geq \mathbf{b}, y \in \mathbb{Z}_+\} \tag{2.15}$$

which is not a polytope. If we treat y as an integer constant, ranging from 0 to 6, we get a 7 disconnected 2-dimensional polytopes, shown in Figure 2.4a.

If we are given an objective $\mathbf{c}$ to minimize, one way of doing this is as follows: apply the simplex algorithm to each of the seven polytopes, then pick the global minimum from the local solutions. The biggest problem with this brute-force approach is that it scales exponentially with the number of integer variables; each feasible combination of integer values gives rise to a polytope that must be explored.



(a) We can show the feasible region in 2 dimensions by treating the integer variable as a constant. Each possible value of the constant gives a (disconnected) part of the feasible region.



(b) If we relax the integrality constraint on the integer variable, the resulting feasible region is a plane (a 2-dimensional polytope) in 3 dimension. This plane contains the feasible region for the unrelaxed problem.

Figure 2.4: Feasible region of a mixed-integer linear program. The red lines (and dots) represent the feasible region. The shaded area represents the relaxed feasible region.

2.2.1 Relaxed Formulation

In the case of (2.14), we are fortunate enough that a much simple approach will yield the optimal solution. We create the *relaxed formulation* for (2.14) by dropping all the integrality constraints. The resulting relaxed feasible region is

$$R_X = \{\mathbf{x} \in \mathbb{R}^3_+ : A\mathbf{x} \geq \mathbf{b}\} \tag{2.16}$$

This feasible region is shown in Figure 2.4b. From the figure, we see that each vertex of R_X is in fact also in X. Since the simplex algorithm by design always finds vertex solutiuons, we can solve (2.14) by ignoring the integrality constraints and feeding the remaining constraints and objective function to simplex. The relaxed version of a MIP or IP is called the *relaxation*, the *linear relaxation* or the *linear programming (LP) relaxation*.

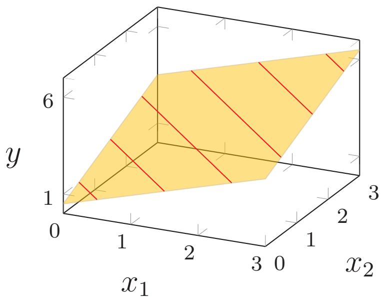
The feasible region for the general MIP (2.4) is

$$X = \{\mathbf{x} \in \mathbb{R}^n_+ \times \mathbf{y} \in \mathbb{Z}^p_+ : A\mathbf{x} + B\mathbf{y} \geq \mathbf{b}\} \tag{2.17}$$

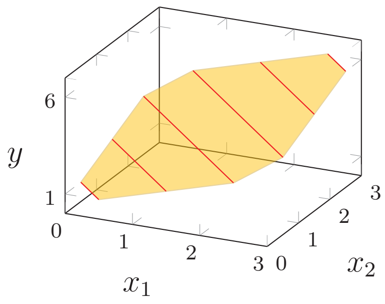
and the relaxed feasible region is

$$R_X = \{\mathbf{x} \in \mathbb{R}^n_+ \times \mathbf{y} \in \mathbb{R}^p_+ : A\mathbf{x} + B\mathbf{y} \geq \mathbf{b}\} \tag{2.18}$$

The mixed-integer program (2.14) was carefully designed so that all the vertexes of its feasible region would take integer values in the integer variable coordinate. In reality, only a very few problems have this very fortunate property.



(a) No vertex of the relaxed feasible region takes an integer value for y.



(b) The perfect reformulation is the convex hull of the original feasible region.

Figure 2.5: We can create a perfect reformulation by cutting away vertices that are not feasible for the original MIP.

2.2.2 Perfect Reformulations

By a very slight modification of (2.14), we get a MIP which can no longer be solved directly using simplex.

$$\begin{aligned} \textbf{minimize} \quad & c_1x_1 + c_2x_2 + fy \\ \textbf{subject to} \quad & x_1 + x_2 - y = -\tfrac{1}{2} \\ & x_1, x_2 \in [0,3] \\ & y \in \mathbb{Z}_+ \end{aligned} \tag{2.19}$$

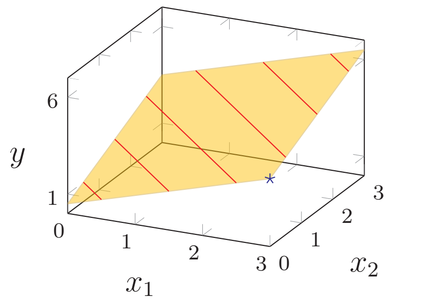
As shown in Figure 2.5a, each vertex of the relaxed feasible region of (2.19) has a fractional value for the integer variable, and is therefore not in the feasible region of (2.19).

One way to resolved this issue by adding *valid inequalities* or *valid cuts* to the formulation. An inequality $\mathbf{a}^T\mathbf{x} \geq a_0$ is valid for a formulation if it holds for each point in the corresponding feasible region. By adding

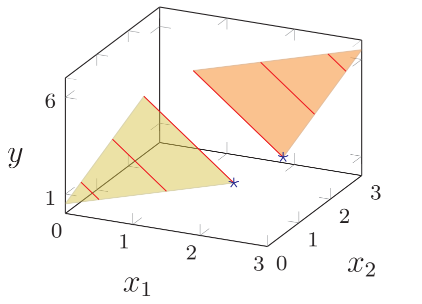
$$\begin{aligned} x_1 + x_2 &\geq 0.5 \\ x_1 + x_2 &\leq 5.5 \\ x_1 - x_2 &\geq -2.5 \\ x_2 - x_1 &\geq -2.5 \end{aligned} \tag{2.20}$$

to (2.19), we get the feasible region shown in Figure 2.5b. By adding (2.20) to (2.19), we have created a *perfect reformulation* of (2.19). In a perfect reformulation of a MIP, each vertex of the relaxed feasible region takes integer values for all integer variables. Alternatively, the feasible region of a perfect reformulation is the convex hull of the original feasible region, i.e. $R_X = \text{conv}(X)$.

If a perfect reformulation is known for a given MIP, then using this reformulation allows us to solve the MIP using simplex. The problem is that



(a) The optimal solution to the relaxed problem is not feasible for the original MIP.



(b) In this case, the relaxed optimal solution in each branch is feasible for the original MIP.

Figure 2.6: When the optimal solution to a relaxed problem has integer variables with fractional solutions, we branch on one of those variables.

finding perfect reformulations to arbitrary MIPs amounts to finding convex hulls, which is, in general, NP-hard.

2.2.3 The Branch-and-Bound Algorithm

When finding perfect reformulations is impractical, we can apply the branch-and-bound algorithm. Given a feasible region X with relaxation R_X, the branch-and-bound algorithm relies on the observation that, since $X \subseteq R_X$,

$$\min_{\mathbf{x}' \in R_X} f(\mathbf{x}') \leq \min_{\mathbf{x}' \in X} f(\mathbf{x}') \leq f(\mathbf{x}) \qquad \forall \mathbf{x} \in X \tag{2.21}$$

That is, feasible solutions provide upper bounds on the optimal value, while optimal solutions to the relaxed problem provide lower bounds on the optimal value.

In the branch-and-bound approach to solving a MIP M, we start by solving the relaxation of M. If the solution has at least one integer variable taking fractional values, we pick one of these variables, say y with fractional value r, and create two new MIPs. In one, we add the constraint $y \geq \lceil r \rceil$, in the other, we add $y \leq \lfloor r \rfloor$. This process is pictured in Figure 2.6 for the MIP (2.19) with the objective $-x_1 + x_2$.

In order to illustrate the branch-and-bound process, we consider the IP

$$\begin{aligned} \textbf{minimize} \quad & 4x + 3y \\ \textbf{subject to} \quad & x + 3y \geq 5 \\ & 12x + 6y \geq 25 \\ & x, y \in \mathbb{Z}_+ \end{aligned} \tag{2.22}$$

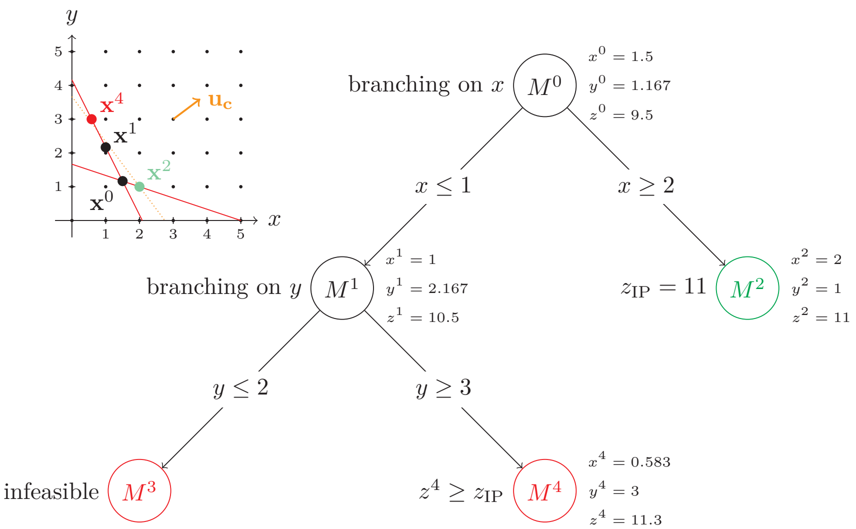


Figure 2.7: The branch-and-bound approach to solving IP (2.22). M_2 gives an integer feasible solution. M_3 is infeasible and no further exploration is required from it. M_4 is not integer feasible, but the LP value is higher than the lower bound established by M_2, so no further exploration is needed. The LP solution to M_2 is the optimal solution.

Figure 2.7 shows the branching tree for (2.22). In the figure, M^0 is the LP relaxation of the IP. When we move down the branching tree, we restrict the feasible region. Thus, the optimal value in the root node of any subtree is a lower bound on the optimal value for any node in that subtree; we cannot improve the solution by limiting the search space.

The idea behind the branch-and-bound approach is to introduce integer bounds on variables taking fractional values until integer solutions are found. At any time, the *incumbent solution* is the best integer solution known in terms of objective value, given that some integer solution has been found.

In the solution presented in Figure 2.7, the first step is to solve M^0. Since the solution is not integer, we branch on the most fractional variable, in this case, x. We may also note z^0 as a lower bound on the optimal value.

Branching on x generates M^1 and M^2. The solution of M_1 is still fractional, but the solution of M^2 is integer. We note $\mathbf{x}^2 = (x^2, y^2)$ as the incumbent solution, with $z_{\text{IP}} = z^4$ as an upper bound on the optimal value. The incumbent solution best known integer solution in terms of objective value.

Since $z^1 < z_{\text{IP}}$, the optimal solution may still be in the subtree rooted at M^1, so we again branch on the most fractional variable. This could be x still, but in this case, we branch on y. The resulting nodes are M^3 and M^4. M^3 is infeasible and can be discarded. The solution of M^4 is fractional, but the optimal value is higher than the existing upper bound, so the subtree rooted at M^4 cannot

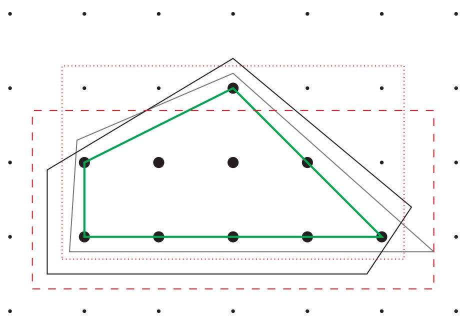


Figure 2.8: Formulations for an integer program. The feasible region is drawn using large, solid dots. The smaller dots are integer points outside the feasible region. The green polygon shows the perfect reformulation (the convex hull) of the feasible region. The black polygon and the gray polygon show other formulations. The red rectangles are not formulations; both rectangles contain infeasible points, and the dashed rectangle does not contain the entire feasible region.

contain an optimal solution, and we can discard it too.

Thus, the optimal solution to (2.22) is $\mathbf{x}^2$ with optimal value z^2. The graph in the upper left of Figure 2.7 shows the solutions to the feasible nodes of the branching tree, together with the objective normalized as $\mathbf{u_c}$. The dotted line through $\mathbf{x}^2$ shows the set of points with the same objective value as $\mathbf{x}^2$.

2.2.4 Formulation Strength

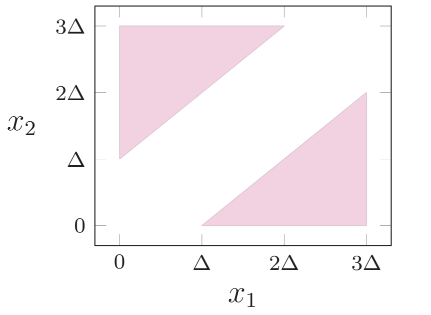
Given a perfect reformulation of a MIP, we can bypass the branch-and-bound process. The solution of the root M^0 of the branching tree would be integer for a perfect reformulation, and the algorithm would perform no branching. Perfect reformulations are an uncommon special case; in general, we look for formulations that lead to branching trees that are as small as possible.

Figure 2.7 shows several formulations for an IP, together with some polygons that are not formulations for the IP. Given two formulations P and Q, we say that P is stronger than Q if $P \subseteq Q$. This is motivated by the observation

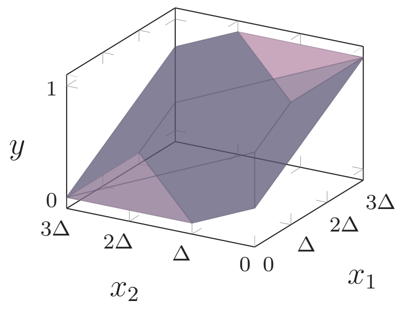
$$P \subseteq Q \qquad \Longrightarrow \qquad \min_{x\in P} f(x) \geq \min_{x\in Q} f(x) \tag{2.23}$$

That is, stronger formulations give tighter bounds on any objective value. Generally, tighter bounds lead to smaller branching trees. This can be seen in Figure 2.7, where we stop branching at M^4 because the lower bound for that node was higher than the incumbent solution.

The two non-perfect formulations in Figure 2.8 are not comparable since neither is a subset of the other. In this case, experimentation would be necessary to determine which formulation performs better in practice. Different MIP formulations for the same problem may use different variables. In this case, a



(a) Non-convex feasible region. The disjunctive constraint divies the feasible region into two disjunct regions.



(b) By introducing a selection variable, we are able to construct a disjunctive constraint polytope.

Figure 2.9: Linearizing a disjunctive constraint by introducing a selection variable.

direct comparison between the formulations will be even more difficult without resorting to experimentation.

2.3 Linearizing Disjunctive Constraints

In continuous-time formulations for scheduling problems, we encounter disjunctive constraints on the continuous variables. This happens, for example, when two vehicles share a resource, and one must pass before the other. We consider the disjunctive constraint

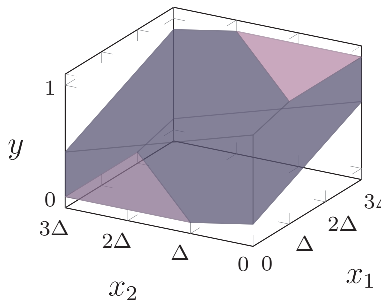
$$x_2 \geq x_1 + \Delta \qquad \text{or} \qquad x_1 \geq x_2 + \Delta \tag{2.24}$$

where $x_1, x_2 \in [0, 3\Delta]$. These constraints result in the fesible region shown in Figure 2.9a.

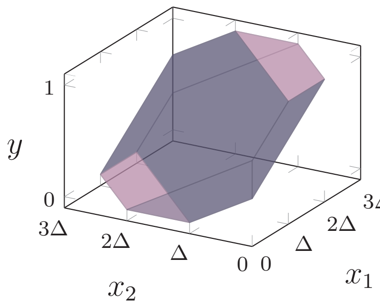
In order to linearize (2.24), we introduce a *selection variable* that will deactivate exactly one of the disjuncts in the constraint. A constraint is *redundant* if it is implied by the rest of the formulation. The big-M trick is a technique for rendering a constraint redundant by adding a very large negative term to one side of the inequality. When we apply the big-M trick to a constraint $\mathbf{a}^T\mathbf{x} \geq b$, we get

$$\begin{aligned} \mathbf{a}^T\mathbf{x} - b &\geq -My \\ y &\in \{0, 1\} \end{aligned} \tag{2.25}$$

where M is a very large number. If the condition for making the constraint redundant is $y = 0$, we replace y with $y - 1$ in (2.25). The big-M trick can be applied whenever a lower bound on $\mathbf{a}^T\mathbf{x} - b$ is implied by the rest of the formulation.



(a) M is too large, which weakens the formulation.



(b) M is too small, which renders the formulation invalid.

Figure 2.10: Getting the right value for M in the big-M trick.

We can apply the big-M trick to (2.24). The result is

$$\begin{aligned} &-x_1 + x_2 + 4\Delta y \geq \Delta \\ &x_1 - x_2 - 4\Delta y \geq -3\Delta \\ \\ &x_1, x_2 \in [0, 3\Delta] \\ &y \in [0, 1] \end{aligned} \tag{2.26}$$

where $M = 4\Delta$, which is just large enough to guarantee redundancy in this case. The feasible region is the polytope shown in Figure 2.9b. The faces defined by $y = 0$ and $y = 1$ correspond, respectively, to the two feasible regions of Figure 2.9a.

If we don't use an appropriate value for M, we end up with one of the situations shown in Figure 2.10. If we set M too large, the resulting formulation is weaker, as shown in Figure 2.10a. If we let M be too small, the resulting formulation cuts away part of the original feasible region, as shown in Figure 2.10b. This formulation is invalid because it cuts away feasible integer solutions.

2.4 Delayed Constraint Generation

The branch-and-bound algorithm presented above is an example of an iterative solution algorithm. In iterative approaches, simplified versions of the problem, typically relaxations, are solved and gradually built up until the solution can be proved feasible (and optimal) for the full problem.

A common iterative approach is *delayed constraint generation*. In this approach, some constraints are dropped from the initial model. As the solution algorithm proceeds, constraints are added back in as needed. For delayed constraint generation to work, we rely on an efficient solution to the corresponding *separation problem*.

Separation problem: Given a set of constraints $\mathcal{A} \subseteq \mathbb{R}^n \times \mathbb{R}$ and a point $\mathbf{x}^* \in \mathbb{R}^n_+$, find a constraint $(\mathbf{a}, b) \in \mathcal{A}$ such that $\mathbf{a}^T\mathbf{x}^* < b$, or determine that no such constraint exists.

When solving a MIP

$$\min\{\mathbf{c}^T\mathbf{x} : A\mathbf{x} \geq \mathbf{b} : \mathbf{x} \in \mathbb{Z}^p_+ \times \mathbb{R}^q_+\} \tag{2.27}$$

we let $\mathcal{A}$ represent the set of constraints $A\mathbf{x} \geq \mathbf{b}$. In each iteration i of the delayed constraint generation process, $\mathcal{A}^i$ is a subset of $\mathcal{A}$ such that

$$\mathcal{A}^0 \subseteq \ldots \subseteq \mathcal{A}^i \subseteq \mathcal{A}^{i+1} \subseteq \ldots \subseteq \mathcal{A} \tag{2.28}$$

Thus, we get a sequence of MIPs

$$\min\{\mathbf{c}^T\mathbf{x} : A^i\mathbf{x} \geq \mathbf{b}^i : \mathbf{x} \in \mathbb{Z}^p_+ \times \mathbb{R}^q_+\} \tag{2.29}$$

with feasible sets Q^i and optimal solutions $(\mathbf{x}^i, z^i)$ respecting

$$\begin{array}{ccccccccccc} Q & \subseteq & \ldots & \subseteq & Q^{i+1} & \subseteq & Q^i & \subseteq & \ldots & \subseteq & Q^0 \\ z & \geq & \ldots & \geq & z^{i+1} & \geq & z^i & \geq & \ldots & \geq & z^0 \end{array} \tag{2.30}$$

If at iteration i, we have $\mathbf{x}^i \in Q$, then $\mathbf{x}^i$ is feasible for the full problem, and thus $z \leq z^i$. Since we already have $z^i \leq z$, we must have $z = z^i$, and $\mathbf{x}^i$ is optimal for the full problem.

If $\mathbf{x}^i \notin Q$, then there must be some $(\mathbf{a}, b) \in \mathcal{A} \setminus \mathcal{A}^i$ for which $\mathbf{a}^T\mathbf{x}^i < b$. Given that we can solve the separation problem, we get $\mathcal{A}^{i+1}$ by adding this constraint, and possibly others, to $\mathcal{A}^i$.

In delayed constraint generation, we are typically working with very large sets $\mathcal{A}$, where only a very few constraints will be violated in each iteration. In some cases, $\mathcal{A}$ is so large that delayed constraint generation is the only practical solution approach. In these cases, we usually rely on an implicit definition of $\mathcal{A}$, which allows us to solve the separation problem without enumeration. In some cases, we may solve a different optimization problem to find the most violated constraint in $\mathcal{A}$.

In the branch-and-bound algorithm, we solve a different type of separation problem. In that case, we look for violated constraints of the form $y_i \in \mathbb{Z}_+$ by simply inspecting each variable, in turn, to see if it takes a fractional value.

2.5 Cutting-Plane Decomposition

When a MIP is too large for a practical solution, we may use a decomposition approach. Here, we will look at one type of cutting-plane decomposition related to delayed constraint generation.

2.5.1 Benders' Decomposition

We consider a MIP on the form

$$\begin{aligned} \textbf{minimize} \quad & \mathbf{c}^T\mathbf{x} + \mathbf{d}^T\mathbf{y} \\ \textbf{subject to} \quad & A\mathbf{x} \geq \mathbf{b} \\ & B\mathbf{x} + D\mathbf{y} \geq \mathbf{g} \\ & \mathbf{x} \in \mathbb{Z}_+^p \times \mathbb{R}_+^q \\ & \mathbf{y} \in \mathbb{R}_+^n \end{aligned} \tag{2.31}$$

That is, $\mathbf{x}$ is a mix of integer and continuous variables, and $\mathbf{y}$ is continuous. This decomposed representation allows the rewriting

$$\begin{aligned} \textbf{minimize} \quad & \mathbf{c}^T\mathbf{x} + f(\mathbf{x}) \\ \textbf{subject to} \quad & A\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}_+^p \times \mathbb{R}_+^q \end{aligned} \tag{2.32}$$

where

$$f(\mathbf{x}) = \min\left\{\mathbf{d}^T\mathbf{y} : B\mathbf{x} + D\mathbf{y} \geq \mathbf{g}, \mathbf{y} \geq \mathbf{0}\right\} \tag{2.33}$$

since $f(\mathbf{x})$ is a (continuous) linear program for a given $\mathbf{x}$, we can write down its dual

$$f'(\mathbf{x}) = \max\left\{(\mathbf{g} - B\mathbf{x})^T\mathbf{u} : D^T\mathbf{u} \leq \mathbf{d}, \mathbf{u} \geq \mathbf{0}\right\} \tag{2.34}$$

Since $f(\mathbf{x}) = f'(\mathbf{x})$ by the strong duality theorem, we can rewrite (2.31) as

$$\begin{aligned} \textbf{minimize} \quad & \mathbf{c}^T\mathbf{x} + \eta \\ \textbf{subject to} \quad & A\mathbf{x} \geq \mathbf{b} \\ & \eta \geq f'(\mathbf{x}) \\ & \mathbf{x} \in \mathbb{Z}_+^p \times \mathbb{R}_+^q \\ & \eta \in \mathbb{R} \end{aligned} \tag{2.35}$$

with the understanding that if $f'(\mathbf{x})$ is unbounded for every $\mathbf{x}$, then so if f, and (2.31) is infeasible.

We note that the feasible region of $f'(\mathbf{x})$ is $\{\mathbf{u} \geq \mathbf{0} : D^T\mathbf{u} \leq \mathbf{d}\}$, which does not depend on $\mathbf{x}$. We let $\mathcal{U}$ be the vertices of this feasible region, and $\mathcal{R}$ be the set of extreme rays. By definition of $f'(\mathbf{x})$, we have

$$f'(\mathbf{x}) \geq (\mathbf{g} - B\mathbf{x})^T\mathbf{u} \qquad \forall \mathbf{u} \in \mathcal{U} \tag{2.36}$$

That is, no vertex can be better than the optimal vertex.

Since $f(\mathbf{x})$, and therefore (2.31), is feasible if and only if $f'(\mathbf{x})$ is bounded, we must also have

$$(\mathbf{g} - B\mathbf{x})^T\mathbf{u} \leq \mathbf{0} \qquad \forall \mathbf{u} \in \mathcal{R} \tag{2.37}$$

That is, $\mathbf{x}$ must be chosen so that no extreme ray makes $f'(\mathbf{x})$ undbounded.

Using these observations about the vertices U and the extreme rays $\mathcal{R}$ of $f'(\mathbf{x})$, we again rewrite (2.31)

$$\begin{array}{rll} \textbf{minimize} & \mathbf{c}^T\mathbf{x} + \eta & \\ \textbf{subject to} & A\mathbf{x} \geq \mathbf{b} & \\ & \eta \geq (\mathbf{g} - B\mathbf{x})^T\mathbf{u} & \mathbf{u} \in \mathcal{U} \\ & 0 \geq (\mathbf{g} - B\mathbf{x})^T\mathbf{u} & \mathbf{u} \in \mathcal{R} \\ & \mathbf{x} \in \mathbb{Z}_+^p \times \mathbb{R}_+^q & \\ & \eta \in \mathbb{R} & \end{array} \tag{2.38}$$

We have now eliminated the variables $\mathbf{y}$, but at the cost of potentially making the problem intractable; we have already seen that $\mathcal{U}$ could be exponential in the number of constraints in D^T.

In order to solve (2.38), we use an approach similar to delayed constraint generation. We start with a master problem which is the relaxed formulation

$$\begin{array}{rl} \textbf{minimize} & \mathbf{c}^T\mathbf{x} + \eta \\ \textbf{subject to} & A\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}_+^p \times \mathbb{R}_+^q \\ & \eta \in \mathbb{R} \end{array} \tag{2.39}$$

Given a solution $\bar{\mathbf{x}}, \bar{\eta}$ to this problem, we solve the subproblem

$$\begin{array}{rl} \textbf{maximize} & (\mathbf{g} - B\bar{\mathbf{x}})^T\mathbf{u} \\ \textbf{subject to} & D^T\mathbf{u} \leq \mathbf{d} \\ & \mathbf{u} \geq \mathbf{0} \end{array} \tag{2.40}$$

If the subproblem is bounded with optimal solution $\mathbf{u}^*$, and $\bar{\eta} < (\mathbf{g} - B\mathbf{x})^T\mathbf{u}^*$ then we add $\eta \geq (\mathbf{g} - B\mathbf{x})^T\mathbf{u}^*$ to the master problem. If the subproblem is bounded with $\bar{\eta} \geq (\mathbf{g} - B\mathbf{x})^T\mathbf{u}^*$, then $\bar{\mathbf{x}}, \bar{\eta}$ is feasible for the subproblem, and an optimal solution has been reached.

If the subproblem is unbounded, then there will be at least one extreme ray $\mathbf{u}'$ for which $(\mathbf{g} - B\bar{\mathbf{x}})^T\mathbf{u}' > 0$, so we add $(\mathbf{g} - B\bar{\mathbf{x}})^T\mathbf{u}' \leq 0$ to the master problem.

2.5.2 Logic-Based Benders' Decomposition

Classical Benders' decomposition (Classic Benders) has a significant advantage in that it provides a general procedure for creating cuts for the master problem. This ability is ensured by the requirement that the subproblem is an LP, as in (2.33).

In logic-based Benders' decomposition (Logic Benders), the subproblem can be any optimization problem, but we must develop the process for creating cuts specifically for each problem. In classic Benders, the dual of the subproblem is used to find lower bounds on the objective of the master problem, as well as feasibility cuts when the subproblem is infeasible for the current master solution. In Logic Benders, an *inference dual* takes the place of the LP dual. Here, we present the Logic Benders setup of Hooker and Ottosson [HO03].

Given an optimization problem

$$\min_{\mathbf{x}\in D}\{f(\mathbf{x}) : \mathbf{x} \in S\} \tag{2.41}$$

where D is the domain of $\mathbf{x}$ and S is the feasible set, the inference dual is the optimization problem

$$\begin{aligned} \textbf{maximize} \quad & \eta \\ \textbf{subject to} \quad & \mathbf{x} \in (S \cap D) \rightarrow f(\mathbf{x}) \geq \eta \end{aligned} \tag{2.42}$$

That is, the solution value to the inference dual is the largest lower bound on the objective function of the primal (2.41) over its feasible region. Thus, the inference dual (2.42) is a strong dual to (2.41) by definition. The problem with using the inference dual is that there is no general solution strategy.

In Logic Benders, we start with a problem of the form

$$\begin{aligned} \textbf{mininize} \quad & f(\mathbf{x}, \mathbf{y}) \\ \textbf{subject to} \quad & (\mathbf{x}, \mathbf{y}) \in S \\ & \mathbf{x} \in D_x \\ & \mathbf{y} \in D_y \end{aligned} \tag{2.43}$$

then, we fix the master variable $\mathbf{x}$ at a *trial value* $\bar{\mathbf{x}}$, which gives us the subproblem

$$\min_{\mathbf{y}\in D_y}\{f(\bar{\mathbf{x}}, \mathbf{y}) : (\bar{\mathbf{x}}, \mathbf{y}) \in S\} \tag{2.44}$$

Letting $S(\bar{\mathbf{x}}) = \{\mathbf{y} : (\bar{\mathbf{x}}, \mathbf{y}) \in S\}$, we get the inference dual

$$\begin{aligned} \textbf{maximize} \quad & \eta \\ \textbf{subject to} \quad & \mathbf{y} \in [S(\bar{\mathbf{x}}) \cap D_y] \rightarrow f(\bar{\mathbf{x}}, \mathbf{y}) \geq \eta \end{aligned} \tag{2.45}$$

The key to Logic Benders is using the solution η^* from the subproblem to create a function $\eta_{\bar{\mathbf{x}}} : D_x \to \mathbb{R}$ such that

$$\begin{aligned} &\eta_{\bar{\mathbf{x}}}(\bar{\mathbf{x}}) = \eta^* \\ &\eta_{\bar{\mathbf{x}}}(\mathbf{x}) \leq f(\mathbf{x}, \mathbf{y}) \qquad (\mathbf{x}, \mathbf{y}) \in S \end{aligned} \tag{2.46}$$

That is, $\eta_{\bar{\mathbf{x}}}$ defines a lower bound on the objective value for any fixed $\mathbf{x}$ and a largest lower bound for $\bar{\mathbf{x}}$.

Based on these functions, we set up an iterative solution process similar to the one in Classic Benders. We let the (iterated) master problem be

$$\begin{array}{llll} & \textbf{minimize} & z & \\ (LB^n) & \textbf{subject to} & z \geq \eta_{\bar{\mathbf{x}}^i}(\mathbf{x}) & i \in \{0, \dots, n-1\} \\ & & z \in \mathbb{R}, \mathbf{x} \in D_x & \end{array} \tag{2.47}$$

For each iteration i, the optimal solution is $(z^i, \bar{\mathbf{x}}^i)$, and the Logic Benders cut generated in the subproblem is $z \geq \eta_{\bar{\mathbf{x}}^i}(\mathbf{x})$.

If we rewrite the Logic Benders master problem as

$$\begin{array}{ll} \textbf{mininize} & z \\ \textbf{subject to} & z \geq f(\mathbf{x}, \mathbf{y}) \\ & (\mathbf{x}, \mathbf{y}) \in S \\ & r \in \mathbb{R} \\ & \mathbf{x} \in D_x \\ & \mathbf{y} \in D_y \end{array} \tag{2.48}$$

we can recognize LB^0 as a relaxation of this problem. Since the Logic Benders cuts are clearly dominated by the constraints $z \geq f(\mathbf{x}, \mathbf{y})$, each LB^i must also be a releaxation. It follows that each z^i is a lower bound on the optimal solution.

When we solve LB^j, we get the solution $(z^j, \bar{\mathbf{x}}^j)$. If the subproblem is feasible for $\bar{\mathbf{x}}^j$, we get the solution $(\eta^j, \bar{\mathbf{y}}^j)$. If $z^j < \eta^j$, then $(z^j, \bar{\mathbf{x}}^j, \bar{\mathbf{y}}^j)$ is not feasible for (2.48), since then $z^j < f(\bar{\mathbf{x}}^j, \bar{\mathbf{y}}^j)$. Therefore, we add the cut $z \geq \eta_{\bar{\mathbf{x}}^j}(\mathbf{x})$.

If, instead, $z^j \geq \eta^j$, the solution $(z^j, \bar{\mathbf{x}}^j, \bar{\mathbf{y}}^j)$ is feasible for (2.48). Then, z^j is both an upper and a lower bound on the objective value, and therefore optimal.

So far, we have assumed the problems and subproblems to be feasible and bounded. We now look at what happens when they are not.

Unbounded Subproblem If the subproblem is unbounded (dual infeasible) at some iteration j, then the full problem is also unbounded. From the unboundedness of subproblem, we get the existene of a sequence $\mathbf{y}_n$ so that

$$\lim_{n \to \infty} f(\bar{\mathbf{x}}, \mathbf{y}_n) = -\infty \tag{2.49}$$

Since $(\bar{\mathbf{x}}, \mathbf{y}_n) \in S$, we let $z_n = f(\bar{\mathbf{x}}, \mathbf{y}_n)$, and so the full problem is unbounded, since each z_n is an upper bound in the optimal solution, and $z_n \to -\infty$.

Infeasible Subproblem If the subproblem is infeasible for some $\bar{\mathbf{x}}$, the dual is unbounded. Thus, the resulting Logic Benders cut should have the property that

$$\mathbf{x} = \bar{\mathbf{x}} \quad \Longrightarrow \quad z \to \infty \tag{2.50}$$

An alternative solution is to replace the standard Logic Benders cut with a custom cut that eliminates any solution for which $\mathbf{x} = \bar{\mathbf{x}}$.

Infeasible Master Problem Each LB^i (2.47) is a relaxation of the full problem (2.48). Thus, if the master problem is infeasible at any time, the full problem is also infeasible.

2.5.2.1 Logic Benders for Scheduling in Sparse Traffic Systems

Logic Benders can be useful in scheduling problems in sparse traffic systems. In such problems, we may initially assume that all traffic can move according to the current plan. By studying candidate solutions, we may identify conflicts that either cause infeasible subproblems or raise the lower bounds on the objective, for example, when a train will be delayed more than the master solution assumed.

If traffic is sparse and conflicts few, then the Logic Benders approach may end up generating very few constraints. However, to use Logic Benders, we require a scheduling formulation with which we can efficiently solve the subproblem.

References

[BT97] Bertsimas, D. and Tsitsiklis, J. N. *Introduction to linear optimization.* Athena Scientific Belmont, MA, 1997.

[HO03] Hooker, J. and Ottosson, G. "Logic-based Benders decomposition". In: *Mathematical Programming* vol. 96, no. 1 (Apr. 2003), pp. 33–60.

[WN99] Wolsey, L. A. and Nemhauser, G. L. *Integer and combinatorial optimization.* Vol. 55. John Wiley & Sons, 1999.

Chapter 3

Contributions

"However beautiful the strategy,
you should occasionally look at the results."

Winston Churchill

3.1 Domain Knowledge and Requirements

Expanding our knowledge of the domains in which we work has been behind many of our new developments. In Paper II, we present workload targets based on discussions with air traffic controllers. In Paper III, we give a detailed description of how to model train scheduling without slack, enabling tighter schedules.

3.1.1 Workload Constraints in Air Traffic Control

At the presentation of Paper I at SESAR Innovations Days 2018, we were able to conduct informal interviews with air traffic controllers in various roles within European centralized air traffic management. Based on these, we created the capacity constraint variants presented in Paper II.

3.1.2 Rail Infrastructure and Business Rules

We have made an extensive study of the infrastructure of Oslo Central Station, including the business rules governing simultaneous train movements in the station. We share our observations and data in Papers III and IV.

3.2 Theory and Modeling

In Papers I and II, we extended the Path&Cycle formulation for the *hotspot problem* in air traffic management. In Paper I, we study different forms of capacity constraints and how these compare in performance. In Paper II, we further expand the flexibility of the capacity constraint modeling, allowing multiple simultaneous constraints. We compare solution quality and performance of different capacity constraint combinations.

In Paper III, we develop a MILP formulation that allows simultaneous dispatching and in-station rerouting. The model also captures many detailed business rules that constrain train movements in a station, allowing very precise scheduling.

3.3 Implementation and Experiments

For Papers II and III, we implemented the solution algorithms from scratch and used the implementations to run extensive computational studies on the performance of the algorithms. In Paper II, we had to craft artificial instances based on observations of traffic and publicly available traffic data. In Paper III, we were able to get detailed infrastructure data from Bane NOR, which allowed us to model Oslo Central Station with high accuracy. We created traffic schedules based on published timetables.

The source code developed for Paper III has been used by SINTEF to create a prototype for real-time dispatching support. This prototype is part of the GOTO project with Bane NOR and is now being tested by dispatchers in Oslo Central Station.

In Paper IV, we share the infrastructure data from the computational experiments in Paper III. This allows other researchers to test their approaches on the same problem.

3.4 Summary of Papers

Paper I introduces new capacity constraints for the alternative-graph based Path&Cycle approach to the Hotspot Problem in Air Traffic Flow Management.

Paper II further expands the Path&Cycle model for Air Traffic Controller workload by showing how we can use layered capacity constraints to provide useful, detailed models for workload restrictions. The paper also gives a detailed presentation of the Path&Cycle solution algorithm.

Paper III shows how to expand alternative-graph-based scheduling approaches also to consider routing. The paper shows that this extended approach can solve the Optimal Dispatching Problem for Oslo Central Station in real use-cases.

Paper IV is a companion paper to Paper III, containing the infrastructure data used for the computational experiments of that paper. We share these data so that other researchers may compare their approaches to ours.

Papers

Paper I

Hotspot Resolution with Sliding Window Capacity Constraints using the Path&Cycle Algorithm

Carlo Mannino, Andreas Nakkerud, Giorgio Sartor, Patrick Schittekat

Published in *SESAR Innovation Days,* December 2018.

Hotspot Resolution with Sliding Window Capacity Constraints using the Path&Cycle Algorithm

Carlo Mannino
SINTEF and
Department of Mathematics,
University of Oslo
Oslo, Norway
carlo.mannino@sintef.no

Andreas Nakkerud
Department of Mathematics,
University of Oslo
Oslo, Norway
andreana@math.uio.no

Giorgio Sartor
SINTEF
Oslo, Norway
giorgio.sartor@sintef.no

Patrick Schittekat
SINTEF
Oslo, Norway
patrick.schittekat@sintef.no

***Abstract*—We extend the new, efficient Path&Cycle formulation for the Hotspot Problem with two methods for dealing with windowed capacity constraints. We also discuss how to combine constraints to allow two-level capacity restricions for peak and average load respectively. Finally, we present computational results for the sliding window capacity constraint.**

I. Introduction

The Hotspot Problem in Air Traffic Management is the problem of avoiding localized congestion in the controlled airspace. The airspace is divided into sectors, each with a capacity constraint. These constraints can limit the number of flights simultaneously in a region, or the number of flights entering a region in given time windows (see Figure 1). A *hotspot* [1], [2] is a sector with a violated capacity constraint. One common approach to eliminating hotspots is to limit the number of flights in fixed time windows. However, this will often result in *bunching* [3], where some of the flights in one window are moved to the beginning of the next, causing the beginnings of all windows to be crowded. In order to combat bunching, we propose to use sliding capacity windows.

In practice, it is not enough to only look at short-term peak capacity. Even when peak capacity is not violated, sustained high load puts too much strain on controllers. We propose combining capacity contraints to allow for both higher-load short-time peak capacity constraints and lower-load long-time capacity constraints. The ability to combine both short-term and long-term of capacity constraints in one model makes for more realistic hotspot resolution.

When checking for capacity violations, we count the number of flights currently in each sector, i.e., occupancy count. When using windowed constraints, another possibility is to count only the entries into the sector during the given window, i.e., entry count. The entry can be a better option if the workload associated with each flight is largely independent of the time that flight spends in the sector. In this paper, we only use occupancy counts, but our algorithms support the use of both counts interchangeably.

The Path&Cycle formulation is a new, efficient formulation for job-shop scheduling problems that was introduced in [4] to tackle the Hotspot Problem in Air Traffic Management. It does not have the disadvantages of time-indexed formulations

1A

2

1B

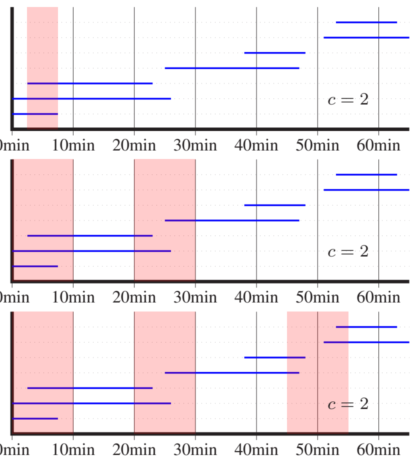


Figure 1. Three types of capacity constraints, each with capacity at most 2. Solid, blue lines represent flights present in this sector. Violations are shown in red. Type 1A violations are instantaneous, Type 2 violations are in fixed windows at 10 minute intervals, Type 1B violations are in any 10 minute interval.

and big-M formulations, which are used in similar approaches. In particular, time-indexed formulations struggle when the number of time periods grow large, while big-M formulations are slowed down due to weak bounds on optimality.

We briefly discuss the Path&Cycle model from [4] in Section II. We also introduce a new generalization of the Path&Cycle model which allows for sliding capacity windows. We discuss how to add a layer of fixed-window capacity constraints to the Path&Cycle model in Section III. Finally, we present computational results for sliding capacity windows in Section IV.

II. The Path&Cycle Model for Type 1 Capacity Constraints

We are solving the Hotspot Problem for a set of sectors S and a set of flights F, under a variety of capacity constraints. A *route node* (f, s) is a pair of a flight and a sector, where the flight f passes through the sector s. Each flight $f \in F$ has an associated route, which is an ordered sequence of route nodes $((f, s_1), (f, s_2), \ldots, (f, s_q))$, where s_1 is the departure sector and s_q is the arrival sector. We let $(f, s+1)$ denote the route node immediately following (f, s). Our goal is to find a schedule $\mathbf{t}$ that specifies for each route node (f, s) the time t_f^s when flight f will enter sector s.

For each route node (f, s), we are also given the time Λ_f^s flight f takes to traverse sector s. For each departure node, we are given the earliest departure time, and for some departure nodes we are also given the latest departure time. In addition to satisfying these time constraints, our schedule must also satisfy the given capacity constraints.

A. *Type 1 Capacity Constraints*

Type 1 capacity constraints are constraints that apply to any interval (or instant) of a given length. Type 1A capacity constraints are violated if there is an instant where the capacity of a sector is exceeded. Type 1B capacity constraints are violated if there is an interval where the number of flights in the sector during that interval is above the capacity of the sector. (See Figure 1.)

The algorithm for Type 1A capacity violations (see Figure 1) from [4] is presented in Section II-B. Type 1B is a generalization of Type 1A, where each flights occupancy in the sector is extended by the length of the desired capacity constraint window (see Figure 2 and Lemma 1). Since Type 1A is Type 1B with zero-width windows, we refer to the more general Type 1B as Type 1.

Lemma 1. *We have a Type 1B violation of capacity c and window width Δ if and only if we have a Type 1A violation of capacity c where each flight occupancy has been extended by Δ after the original occupancy.*

Proof: Suppose we have a Type 1B violation of capacity c and window width Δ. Let t be the time at the end of the window. Now extend the duration of each flight by Δ. Each flight present in the window will now either be present at t, or have its extension present at t, so we have a Type 1A violation at time t.

Suppose, for the other direction, that we have extended the durations of all flights by Δ, and that we have a Type 1A violation of capacity c at time t. If we remove each extension, then all the flights with an extension present at t must have been present between $t - \Delta$ and t, so we have a Type 1B violation of capacity c and window width Δ. ■

B. *The Path&Cycle Formulation for Type 1A*

We first develop the algorithms and models needed to solve for Type 1A capacity constraints. We then describe the generalization to Type 1B in Section II-D.

Let t_f^s be the time flight f enters sector s, and let Λ_f^s be the time flight f takes to traverse sector s. When the flight is understood from context, we use $s+1$ to denote the next sector in the flight's path. We get the equality

$$t_f^{s+1} = t_f^s + \Lambda_f^s, \tag{1}$$

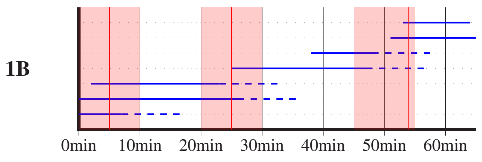


Figure 2. By extending the duration of each flight's occupation of the region, we can turn 1B violations into 1A violations. We use dashed lines to represent occupancy extensions.

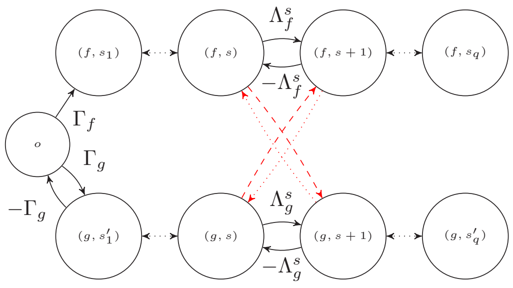


Figure 3. Route node graph. The black edges between route nodes of a single flight represent departure constraints and sector traversal times. The red edges between route nodes on different flights (conflict edges) represent scheduling constraints between flights. In this example f has an earliest departure time, while g has a fixed departure time.

which we will represent using the inequalities $t_f^{s+1} \geq t_f^s + \Lambda_f^s$ and $t_f^s \geq t_f^{s+1} - \Lambda_f^s$.

The earliest departure time, relative to the reference time t_o, of a flight f is denoted Γ_f. Thus

$$t_f^{s_1} \geq t_o + \Gamma_f. \tag{2}$$

If the flight must depart on schedule, then we also add the reverse inequality to make an equality. In our problem, we assume flights cannot be delayed in the air. Therefore, flights arriving from outside the managed area, and flights already in the air, are accounted for using fixed departure times.

We represent (1) and (2) using a weighted, directed graph where the nodes are route nodes, and the weighted, directed edges represent inequalities. Figure 3 shows the resulting graph.

When there is an edge from (f, s) to (f', s'), with weight w, this means that $t_{f'}^{s'} \geq t_f^s + w$. The red, diagonal edges between route nodes of different flights in Figure 3 represent possible *conflict edges*. The dashed edges (going left to right) together represent a meeting of f and g in s. The two constraints together require that both flights enter s before either leaves. The dotted edges (right to left) each represent a precedence constraint. In each case, one flight has to leave s before the other enters. See [5] for more details on disjunctive graphs.

Associated to the conflict edges, we introduce the variables

$$x^s_{fg} = \begin{cases} 1 & \text{if } f \text{ and } g \text{ meet in } s, \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

$$y^s_{fg} = \begin{cases} 1 & \text{if } f \text{ precedes } g \text{ in } s, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Note that for any pair of flights f, g, and for any region s, we must have

$$y^s_{fg} + y^s_{gf} + x^s_{fg} = 1. \tag{5}$$

We let $G(\mathbf{y}, \mathbf{x})$ be the route node graph with added conflict edges such that $\mathbf{x}$ and $\mathbf{y}$ are incidence matrices. We divide the set K of all conflict edges into the two sets K_x and K_y. Note that K, K_x, and K_y contains all conflict edges of the matching type, not only those currently selected.

If $y^s_{fg} = 1$, we add the inequality

$$t^{s+1}_f \leq t^s_g, \tag{6}$$

that is, f leaves s before g enters. If $x^s_{fg} = 1$, we add the pair of inequalities

$$t^{s+1}_f \geq t^s_g \tag{7}$$

$$t^{s+1}_g \geq t^s_f, \tag{8}$$

that is, neither may leave s before both have arrived.

Lemma 2. *If $G(\mathbf{y}, \mathbf{x})$ contains a strictly positive directed cycle, then the set of inequalities corresponding to the edges of the cycle is inconsistent (infeasible).*

Proof: If $G(\mathbf{y}, \mathbf{x})$ contains a directed cycle visiting node (f, s), and the sum total weight of the edges in the cycle is $W > 0$, then

$$t^s_f \geq t^s_f + W > t^s_f.$$

■

From Lemma 2, it follows that $G(\mathbf{y}, \mathbf{x})$ cannot contain strictly positive directed cycles. This restricts the possible values of $\mathbf{x}$ and $\mathbf{y}$.

Given a G with no strictly positive directed cycles, we can always find a longest path from o to any route node $u = (f, s)$. We label this distance $L^*(\mathbf{y}, \mathbf{x}, u)$.

Lemma 3. *Let $\mathbf{x}$ and $\mathbf{y}$ be such that $G(\mathbf{y}, \mathbf{x})$ does not contain any strictly positive directed cycles, let f be a flight, and s be a sector. Minimizing t^s_f subject to the instances of* (6), (7), *and* (8) *corresponding to conflict edges in $G(\mathbf{y}, \mathbf{x})$ is equivalent to minimizing $L^*(\mathbf{y}, \mathbf{x}, (f, s))$.*

Proof: By the same argument as in the proof of Lemma 2, we have that

$$t_u \geq t_o + L^*(\mathbf{y}, \mathbf{x}, u).$$

This inequality is most restricting, since L^* is the longest path. Therefore, when minimizing t^s_f, we can instead minimize $L^*(\mathbf{y}, \mathbf{x}, (f, s))$. ■

Lemma 3 is the key to building a model with no direct reference to the scheduling variables. All connections to time are coded into the edge weights of the route node graph.

C. The Mixed-Integer Linear Programming Model

The variables for our model are $\mathbf{x}$ and $\mathbf{y}$, introduced in Section II-B. For each sector, these variables encode meetings of flights, and precedence between flights that do not meet.

Lemmas 2 and 3 give us most of what we need to build our model. All the scheduling and conflict inequalities are encoded in the graph $G(\mathbf{y}, \mathbf{x})$. The only thing missing is the encoding of capacity constraints.

Lemma 4. *Let F be the set of all flights, s a sector, and c_s the capacity of s. The capacity constraint c_s is at all times respected if and only if for all $\bar{F} \subseteq F$ where $|\bar{F}| = c_s + 1$, we have*

$$\sum_{\{f,g\} \subseteq \bar{F}} x^s_{fg} \leq \binom{|\bar{F}|}{2} - 1. \tag{9}$$

Proof: Suppose the capacity is violated at some point in time. At that time, at least $c_s + 1$ flights must be in s. Let $\bar{F}$ contain any $c_s + 1$ of these flights. The number of pairs in $\bar{F}$ is $\binom{|\bar{F}|}{2}$. Since each pair is meeting, the sum in (9) is $\binom{|\bar{F}|}{2}$, and so the inequality is violated.

Conversely, suppose (9) is violated. Then there is a set $\bar{F}$ of $c_s + 1$ flights with at least $\binom{|\bar{F}|}{2}$ pairwise meetings. Since this is the total number of possible meetings, the flights must all meet. Since they all meet, none may leave s before all have entered, and so there is a point in time when they are all present, and the capacity is violated. ■

Lemma 5. *Let $\mathcal{C}$ be the set of all strictly positive directed cycles in $G(\mathbf{1}, \mathbf{1})$. $G(\mathbf{y}, \mathbf{x})$ contains the strictly positive directed cycle $C \in \mathcal{C}$ if and only if*

$$\sum_{e \in C \cap K_y} y_e + \sum_{e \in C \cap K_x} x_e = |C \cap K|.$$

Proof: C is a subgraph of $G(\mathbf{y}, \mathbf{x})$ exactly when all conflict edges in C, numbering $|C \cap K|$, are selected. ■

One typical objective is to minimize the sum of delays of all flights. This is equivalent to minimizing the sum of arrival times, since our scheduling constraints make it impossible to schedule early arrivals. If we let A be the set of arrival nodes, then our goal is to minimize $\sum_{u \in A} t_u$. Using the results of Section II-B and Lemmas 4 and 5, we get the following Linear Programming model.

$$\begin{array}{rll} \textbf{min} & \sum_{u \in A} L^*(\mathbf{y}, \mathbf{x}, u) & \\ \textbf{s.t.} & & \\ (i) & y^s_{fg} + y^s_{gf} + x^s_{fg} = 1, & \{f, g\} \subseteq F, s \in S, \\ (ii) & \sum_{e \in C \cap K_y} y_e + \sum_{e \in C \cap K_x} x_e \leq |C \cap K| - 1, & \\ & & C \in \mathcal{C}, \\ (iii) & \sum_{\{f,g\} \subseteq \bar{F}} x^s_{fg} \leq \binom{|\bar{F}|}{2} - 1, & \\ & & s \in S, \bar{F} \subseteq F, |\bar{F}| = c_s + 1, \\ & \mathbf{y} \in \{0,1\}^{|K_y|}, \mathbf{x} \in \{0,1\}^{|K_x|}. & \end{array} \tag{10}$$

D. Generalization to Type 1B

By Lemma 1 (see also Figure 2), we know that we model Type 1B constraints by modifying (6), (7), and (8). We use Δ to denote the width of our Type 1B capacity windows. For $y^s_{fg} = 1$, we get

$$t^{s+1}_f + \Delta \leq t^s_g, \tag{11}$$

and for x^s_{fg} we get

$$t^{s+1}_f + \Delta \geq t^s_g \tag{12}$$

$$t^{s+1}_g + \Delta \geq t^s_f. \tag{13}$$

The only required change is modification of weights in the route node graph G.

E. Row and Column Generation

Model (10) is well suited to delay row and column generation. First, we need only generate $y^s_{fg}, y^s_{gf}, x^s_{fg}$ and the associated row of type (10.i) if f and g are violating a capacity constraint.

Of the constraints of type (10.ii) and (10.iii), most rows will not be relevant. Further, in the relevant rows, we do not need to worry about ungenerated variables, as we always generate the x's that may take value 1, i.e., those that represent flights that may meet in the given sector.

Dealing with the objective function is the challenging aspect of delayed row and column generation in this model. This is because the longest path from o to any $u \in A$ will depend on the choice of $\mathbf{x}$ and $\mathbf{y}$ through $G(\mathbf{y}, \mathbf{x})$.

Let $\mathcal{H}$ be the set off all $G(\mathbf{y}, \mathbf{x})$, such that $\mathbf{x}$ and $\mathbf{y}$ satisfy (10.i), (10.ii), and (10.iii). We use $P_u(H)$ to denote (the set of edges of) a longest path from o to u in H for $u \in A$ and $H \in \mathcal{H}$. $L_u(H)$ is the length of $P_u(H)$. If all the conflict edges in H are chosen by the current solution, then $L_u(H) = L^*(\mathbf{y}, \mathbf{x}, u)$.

If all the conflict edges of H are selected, then

$$\sum_{e \in P_u(H) \cap K_x} x_e + \sum_{e \in P_u(H) \cap K_y} y_e = |K \cap P_u(H)|. \tag{14}$$

That is, the set of inequalities

$$L_u(H) \left(\sum_{e \in P_u(H) \cap K_x} x_e + \sum_{e \in P_u(H) \cap K_y} y_e - |K \cap P_u(H)| + 1 \right) \leq \mu_u, \qquad H \in \mathcal{H}, \tag{15}$$

is equivalent to

$$L^*(\mathbf{y}, \mathbf{x}, u) \leq \mu_u. \tag{16}$$

Thus, by expressing the objective of (10) in terms of μ_u and adding the inequalities (15), we obtain the Path&Cycle formulation. When solving this model, we can start with $\mathcal{H} = \mathcal{C} = \emptyset$, and only add inequalities for longest paths (H) and cycles (C) when they become relevant. The steps of the delayed row and column generation algorithm are described in detail in [4].

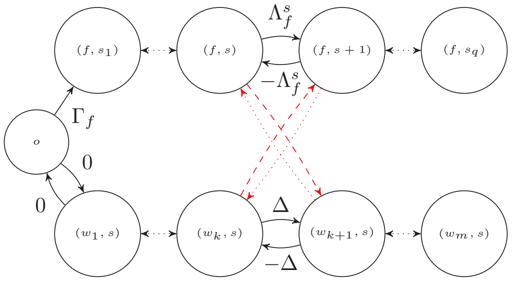


Figure 4. The figure shows how we can adapt the route node graph to model Type 2 capacity constraints. The lower line of nodes represent capacity windows for a single sector s, which is visited by flight f.

III. Modelling Type 2 Capacity Windows

We now present a way to add Type 2 capacity constraints to our current model. This allows us to use two different, simultaneous capacity constraints.

We adapt existing methods in order to model Type 2 capacity constraints. Figure 4 shows how we modify the route node graph in order to account for capacity windows. We can make a further simplification by linking the window nodes directly to the origin o (see Figure 5).

For each sector s and each time window w, we introduce a new *window node* (w, s). We extend $\mathbf{x}$ and $\mathbf{y}$ by thinking of each w as a flight. This means that

$$x^s_{fw} = \begin{cases} 1 & f \text{ is in } s \text{ in window } w, \\ 0 & \text{otherwise,} \end{cases} \tag{17}$$

$$y^s_{fw} = \begin{cases} 1 & f \text{ leaves } s \text{ before window } w \text{ begins,} \\ 0 & \text{otherwise,} \end{cases} \tag{18}$$

$$y^s_{wf} = \begin{cases} 1 & f \text{ enters } s \text{ after window } w \text{ ends,} \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

We use t_w to denote the start time of window w, the end time is then $t_w + \Delta$. We let m be the number of windows, and label the windows $w_1, \ldots, w_m$. For each $i \leq m$ and for each sector s, we add the window node (w_i, s) to the route node graph G. We also add, for each $i \leq m$, edges corresponding to the pair of inequalities

$$t^s_{w_i} = t_o + (i-1)\Delta. \tag{20}$$

That is, and edge from o to (w_i, s) with weight $(i-1)\Delta$, and an edge from (w_i, s) to o with weight $-(i-1)\Delta$.

If $x^s_{fw} = 1$, we add the inequalities (and corresponding edges)

$$t^s_f \leq t^s_w + \Delta \tag{21}$$

$$t^{s+1}_f \geq t^s_w, \tag{22}$$

if $y^s_{fw} = 1$, we add the inequality

$$t^{s+1}_f \leq t^s_w, \tag{23}$$

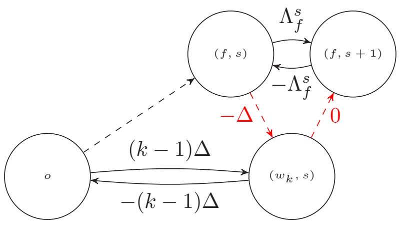


Figure 5. Sub graph of the route node graph. This subgraph encodes the fact that flight f is in sector s in window w_k.

and if $y^s_{wf} = 1$, we add

$$t^s_f \geq t^s_w + \Delta. \tag{24}$$

We are assuming, without loss of generality, a fixed window size Δ. In order to use a variable window size Δ^s_w, simply modify (20) as follows

$$t^s_{w_i} = t_o + \sum_{j=1}^{i-1} \Delta^s_{w_j}, \tag{25}$$

and replace Δ with Δ^s_w in (21) and (24). Note that Δ^s_w is free to vary by window and sector, so that each sector can use its own set of windows.

Figure 5 shows the edges added to the route node graph when flight f is in region s in window w_k. The edges between o and (w_k, s) represent (20), the edges between the window node and the route nodes represent (21) and (22).

So far, we have extended the route node graph $G(\mathbf{y}, \mathbf{x})$ in order to account for the scheduling constraints for capacity windows. What remains is to define proper constraints on the new variables in $\mathbf{x}$ and $\mathbf{y}$. We let W be the set of windows. As before, we have

$$y^s_{fw} + y^s_{wf} + x^s_{fw} = 1, \qquad f \in F, w \in W, s \in S. \tag{26}$$

The added capacity constraint is simpler, since we now only need to count the number of flights that appear in each window. We let c^w_s be the capacity for window w and sector s, then

$$\sum_{f\in F} x^s_{fw} \leq c^w_s, \qquad s \in S, w \in W. \tag{27}$$

By adding these new inequalities to (10), we get the following model.

$$\begin{array}{lll}
\textbf{min} & \sum_{u\in A} L^*(\mathbf{y}, \mathbf{x}, u) & \\
\textbf{s.t.} & & \\
(i.a) & y^s_{fg} + y^s_{gf} + x^s_{fg} = 1, & \{f, g\} \subseteq F, s \in S, \\
(i.b) & y^s_{fw} + y^s_{wf} + x^s_{fw} = 1, & f \in F, w \in W, s \in S, \\
(ii) & \sum_{e\in C\cap K_y} y_e + \sum_{e\in C\cap K_x} x_e \leq |C \cap K| - 1, & \\
 & & C \in \mathcal{C}, \\
(iii.a) & \sum_{\{f,g\}\subseteq \bar{F}} x^s_{fg} \leq \binom{|\bar{F}|}{2} - 1, & \\
 & & s \in S, \bar{F} \subseteq F, |\bar{F}| = c_s + 1, \\
(iii.b) & \sum_{f\in F} x^s_{fw} \leq c^w_s & s \in S, w \in W, \\
 & \mathbf{y} \in \{0, 1\}^{|K_y|}, \mathbf{x} \in \{0, 1\}^{|K_x|}. &
\end{array} \tag{28}$$

This model is suited for the same delayed row and column generation as in Section II-E.

IV. Computational Results

Tables I and II show the results of our experiments with Type 1 capacity constraints, based on simulated data. The first columns of Table I shows the performance of our Type 1A algorithm. The running times are longer in the cases with Type 1B capacity windows of 10 seconds and 1 minute, but this is expected; there are more hotspots when we use time windows, as shown in Figure 1. The data in Table I also confirms that the algorithms have to resolve more hotspots for the wider windows.

In our experiments, the Path&Cycle formulation solves all test instances within a few seconds, while the standard Big-M formulation times out at 10 minutes on some of the more difficult instances, especially with the longer 1 minute windows.

In Table II, we show results from the same instances as in Table I with the same capacity (left-most colums), but using 10 minute Type 1B windows. In this case, the number of hotspots grew too large and almost all of the computations timed out at 10 minute limit. In the worst instances, over 100 hotspots were resolved before time ran out.

We have designed our instances to have a reasonable amount of hotspots with Type 1A capacity constraints. For a proper test of the Type 1B constraints, we need to increase the capacity or change the instances to reduce the number of hotspots. In Table II, we also show the effect of increasing the capacity of the sectors. As the capacity increases, the number of hotspots go down, and many more instances are solved within the time limit.

Our experiments were done with a C# implementation using CPLEX 12.8. CPLEX was set to default parameters, except the number of available threads were set to 1, the advanced start switch was set to 0, and both dual reduction and dynamic search were disabled. The code was run on an Intel i7-7700 HQ 2.8 GHz CPU, with 32 GB of RAM.

TABLE I
COMPUTATIONAL RESULTS FOR TYPE 1A AND TYPE 1B CAPACITY CONSTRAINTS. PATH&CYCLE RESULTS ARE LABELED PC, BIG-M FORMULATION RESULTS ARE LABELED BM. TIME LIMIT WAS SET TO 600 SECONDS, AND SOME BM COMPUTATIONS TIMED OUT. THE NUMBER OF HOTSPOTS INCREASES WITH THE WIDTH OF THE CAPACITY WINDOWS, THIS IN TURN INCREASES COMPUTATION TIME. THE "NODES" COLUMNS SHOW HOW MANY NODES WERE VISITED BY THE BRANCH AND BOUND ALGORITHM USED BY THE MILP SOLVER IN CPLEX.

$\|F\|$	c_s	**Type 1A**						**Type 1B, 10 seconds time window**						**Type 1B, 1 minute time window**					
		Hotspots		**Nodes**		**Time (s)**		**Hotspots**		**Nodes**		**Time (s)**		**Hotspots**		**Nodes**		**Time (s)**	
		PC	BM	PC	BM	PC	BM	PC	BM	PC	BM	PC	BM	PC	BM	PC	BM	PC	BM
122	3	16	15	1e+3	9e+4	1.24	10.99	17	16	2e+3	9e+5	1.45	168.4	20	20	2e+4	3e+6	4.16	449.3
137	3	21	21	5e+3	2e+6	2.24	117.9	21	21	7e+3	1e+7	2.17	494.3	19	15	7e+3	3e+6	2.22	—
131	3	12	12	3e+2	3e+4	0.48	2.99	12	12	3e+2	5e+4	0.49	4.79	13	13	1e+3	3e+4	0.62	2.97
142	3	13	13	5e+2	1e+5	0.53	29.93	13	13	9e+2	1e+5	0.8	26.34	17	17	3e+3	8e+5	1.99	118.5
110	3	12	12	5e+2	4e+4	0.29	9.82	12	12	5e+2	5e+4	0.4	11.55	16	16	9e+3	8e+5	1.58	113.7
127	3	11	11	2e+2	1e+4	0.35	1.04	12	12	3e+2	9e+3	0.52	0.89	18	18	2e+3	2e+4	1.09	1.87
115	3	1	1	0e+0	7e+0	0.05	0.04	1	1	0e+0	6e+0	0.04	0.04	1	1	0e+0	7e+0	0.04	0.04
120	3	4	4	8e+0	1e+1	0.04	0.07	4	4	3e+0	5e+1	0.04	0.06	5	5	9e+0	5e+1	0.04	0.07
131	3	7	7	3e+1	2e+3	0.12	0.36	7	7	3e+1	2e+3	0.13	0.34	8	8	5e+1	5e+3	0.12	0.78
143	3	8	8	6e+1	2e+3	0.14	0.46	8	8	2e+1	2e+3	0.14	0.75	10	10	2e+2	3e+4	0.2	6.42
136	3	15	15	5e+2	6e+4	0.29	17.21	17	17	4e+2	2e+6	0.28	—	20	20	2e+3	2e+6	0.63	—
142	3	9	9	2e+2	6e+3	0.1	1.68	9	9	3e+2	5e+3	0.16	1.41	12	12	1e+3	1e+5	0.45	14.18
139	3	14	14	1e+2	2e+4	0.27	4.42	14	14	1e+2	7e+4	0.29	8.83	20	20	8e+2	6e+5	0.73	76.77
126	3	10	10	2e+2	7e+3	0.24	1.38	11	11	4e+2	1e+4	0.33	1.9	15	14	2e+3	3e+5	0.77	49.75
139	3	19	19	1e+4	2e+5	2.16	31.16	19	19	1e+4	1e+6	1.99	266.7	24	24	3e+4	1e+6	5.29	134.9
288	5	8	8	8e+1	7e+3	0.54	1.52	8	8	7e+1	5e+3	0.51	1.24	12	12	6e+2	1e+5	1.32	18.41
289	5	9	9	3e+1	2e+4	0.23	7.31	10	10	7e+1	2e+4	0.26	9.41	14	14	3e+3	2e+6	1.95	—
278	5	10	10	4e+2	4e+4	0.92	9.88	11	11	5e+2	5e+4	1	12.29	16	14	7e+3	2e+6	3.74	—
259	5	3	3	0e+0	5e+2	0.1	0.23	3	3	0e+0	4e+2	0.11	0.2	6	6	2e+1	8e+2	0.15	0.44
254	5	8	8	7e+1	7e+3	0.31	2.09	8	8	7e+1	8e+3	0.41	2.1	9	9	6e+2	1e+5	0.58	22.85
279	5	9	9	5e+2	4e+4	0.66	8.37	9	9	8e+2	1e+4	0.82	3.91	14	14	5e+3	2e+6	2.38	—
287	5	3	3	0e+0	4e+2	0.12	0.24	6	6	0e+0	2e+3	0.33	0.57	10	10	6e+1	4e+3	0.36	1.78
259	5	11	11	8e+1	1e+4	0.59	2.52	14	14	1e+2	4e+4	0.86	8.96	16	16	7e+2	7e+5	1.63	136.3
281	5	8	8	2e+2	9e+3	0.75	1.85	9	9	2e+2	2e+4	0.83	3.83	12	12	5e+2	8e+4	1.28	23.86
296	5	4	4	4e+1	1e+3	0.16	0.67	4	4	4e+1	2e+3	0.16	0.93	6	6	2e+2	8e+3	0.5	1.8
275	5	7	7	2e+1	8e+2	0.24	0.69	7	7	2e+1	3e+3	0.25	1.46	8	8	2e+2	5e+4	0.39	16.78
256	5	5	5	2e+2	3e+3	0.37	0.83	5	5	2e+2	3e+3	0.4	0.78	7	7	7e+2	3e+3	0.7	1
273	5	9	9	2e+2	1e+4	0.6	3.14	9	9	3e+2	4e+4	0.69	12.37	12	12	2e+3	2e+5	1.92	32.34
274	5	9	9	7e+1	2e+4	0.67	4	9	9	5e+2	2e+4	0.84	4.77	16	16	5e+3	3e+6	3.37	—
287	5	11	11	1e+3	2e+4	0.89	5.34	12	12	1e+3	4e+4	1.48	9.93	16	16	5e+3	1e+6	4.18	193.5

V. CONCLUSIONS

We have shown (see Section IV) that our algorithm for Type 1A (see [4]) also efficiently solves for Type 1B capacity constraints with short time windows, and that it performs well compared to the standard Big-M formulation on almost all our instances.

With a large number of flights, low capacities, and long capacity windows, the number of hotspots becomes too large to handle. However, according to the results in Tables I and II, the Path&Cycle formulation can easily handle up to 20–30 hotspots (including those introduced by intermediate solutions).

In order to further prove our model, we need access to real instances. While we have shown that our model performs well compared to one established approach, we still need to confirm that it performs well in real life.

VI. FUTURE WORK

Our next step is to integrate Type 2 constraints into our model, so that we can solve the Hotspot Problem with layered capacity constraints. This will allow us to restrict, simultaneously, peak and average load. By tuning window sizes, it is possible to approximate a wide range of capacity constraint schemes. We will also introduce entry counts as an alternative to occupancy counts for defining capacity constraints.

Using this model in practice would allow for more realistic modeling of the air traffic controller's workload capacity contraints, and therefore result in a more achievable work load. Also, the use of sliding windows have the added benefit of reducing the drive towards bunching.

Furthermore, our experiments have indicated that the Path&Cycle algorithm is well suited to reoptimization with slight variations in the model. This makes the algorithm ideal for the approach to inter-airline scheduling fairness presented by Jacquillat and Vaze [6]. We will include this approach to fairness in our future models, so that we can evaluate the fairness criteria themselves in terms of performance and effectiveness.

TABLE II
COMPUTATIONAL RESULTS FOR TYPE 1B CAPACITY CONSTRAINTS WITH 10 MINUTE WINDOWS. PATH&CYCLE RESULTS ARE LABELED PC, BIG-M FORMULATION RESULTS ARE LABELED BM. TIME LIMIT WAS SET TO 600 SECONDS. AT THE LOWER CAPACITIES, THE NUMBER OF HOTSPOTS GROW VERY LARGE, AND ALMOST ALL COMPUTATIONS TIME OUT. AS THE CAPACITY INCREASES, THE NUMBER OF HOTSPOTS BECOMES MANAGEABLE.

	Type 1B, 10 minutes time window					**Type 1B, 10 minutes time window**					**Type 1B, 10 minutes time window**				
$\|F\|$	c_s	**Hotspots**		**Time (s)**		c_s	**Hotspots**		**Time (s)**		c_s	**Hotspots**		**Time (s)**	
		PC	BM	PC	BM		PC	BM	PC	BM		PC	BM	PC	BM
122	3	61	47	—	—	4	11	11	1.49	61.2	5	1	1	0.19	0.07
137	3	75	58	—	—	4	16	16	1.68	47.78	5	1	1	0.07	0.06
131	3	52	47	—	—	4	5	5	0.1	0.18	5	0	0	0.03	0.04
142	3	52	48	—	—	4	8	8	0.13	0.28	5	0	0	0.02	0.03
110	3	47	47	—	—	4	4	4	0.06	0.32	5	0	0	0.02	0.02
127	3	58	63	—	—	4	19	19	4.57	74.73	5	2	2	0.06	0.08
115	3	14	14	0.2	53.24	4	0	0	0.02	0.03	5	0	0	0.02	0.03
120	3	33	32	43.76	—	4	1	1	0.05	0.04	5	0	0	0.03	0.02
131	3	50	35	—	—	4	6	6	0.11	0.47	5	0	0	0.03	0.04
143	3	51	54	—	—	4	2	2	0.04	0.07	5	0	0	0.03	0.03
136	3	65	62	—	—	4	7	7	0.36	8.25	5	0	0	0.02	0.03
142	3	54	53	—	—	4	9	9	0.79	24.43	5	0	0	0.03	0.03
139	3	63	50	—	—	4	9	9	0.58	2.68	5	1	1	0.05	0.06
126	3	68	71	—	—	4	16	16	1.22	5.46	5	2	2	0.05	0.08
139	3	76	56	—	—	4	21	21	8.19	—	5	3	3	0.04	0.06
288	5	59	60	—	—	6	31	24	39.06	—	7	2	2	0.18	0.26
289	5	102	103	—	—	6	46	36	—	—	7	7	7	0.29	1.32
278	5	0	142	—	—	6	48	41	—	—	7	4	4	0.2	0.52
259	5	57	58	—	—	6	37	25	—	—	7	7	7	0.49	5.05
254	5	44	44	—	—	6	25	14	53.25	—	7	6	6	0.79	0.82
279	5	75	76	—	—	6	29	25	—	—	7	9	9	0.47	1.47
287	5	84	83	—	—	6	42	27	—	—	7	4	4	0.22	0.7
259	5	87	86	—	—	6	27	20	487.6	—	7	2	2	0.14	0.21
281	5	91	89	—	—	6	43	31	—	—	7	11	11	2.57	5.07
296	5	60	62	—	—	6	24	21	213.2	—	7	2	2	0.12	0.24
275	5	68	67	—	—	6	16	16	18.31	—	7	1	1	0.11	0.13
256	5	56	57	—	—	6	32	19	—	—	7	6	6	0.39	1.14
273	5	0	97	—	—	6	48	29	—	—	7	10	10	2.72	8.05
274	5	93	93	—	—	6	44	37	—	—	7	10	10	1.41	—
287	5	0	112	—	—	6	36	24	—	—	7	8	8	1.24	14.26

REFERENCES

[1] C. Allignol, N. Barnier, P. Flener, and J. Pearson, "Constraint programming for air traffic management: a survey: In memory of pascal brisset," *The Knowledge Engineering Review*, vol. 27, no. 3, p. 361392, 2012.

[2] T. Dubot, J. Bedouet, and S. Degrmont, "Modelling, generating and evaluating sector configuration plans," in *30th Congress of the International Council of the Aeronautical Sciences (ICAS 2016)*, 2016.

[3] S. Stoltz and P. Ky, "Reducing traffic bunching through a more flexible air traffic flow management," in *4th USA/R&D Seminar*, 2001.

[4] C. Mannino and G. Sartor, "The Path&Cycle Formulation for the Hotspot Problem in Air Traffic Management," in *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*, ser. OpenAccess Series in Informatics (OASIcs), R. Borndörfer and S. Storandt, Eds., vol. 65. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 14:1–14:11. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2018/9719

[5] A. Mascis and D. Pacciarelli, "Job-shop scheduling with blocking and no-wait constraints," *European Journal of Operational Research*, vol. 143, no. 3, pp. 498 – 517, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221701003381

[6] A. Jacquillat and V. Vaze, "Interairline equity in airport scheduling interventions," *Transportation Science*, vol. 52, no. 4, 2018.

Paper II

Air Traffic Flow Management with Layered Workload Constraints

Carlo Mannino, Andreas Nakkerud, Giorgio Sartor

Published in *Computers & Operations Research,* November 2020.

Computers and Operations Research 127 (2021) 105159

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/caor

Air traffic flow management with layered workload constraints [☆]

Carlo Mannino [a,b], Andreas Nakkerud [a,b,*], Giorgio Sartor [b]

[a] *Department of Mathematics, P.O box 1053 Blindern, 0316 Oslo, Norway*
[b] *SINTEF, P.O box 124 Blindern, 0314 Oslo, Norway*

A R T I C L E   I N F O

Article history:
Received 4 May 2020
Revised 17 November 2020
Accepted 20 November 2020
Available online 28 November 2020

2010 MSC:
90B06
90B20
90C06
90C08
90C11
90C90

Keywords:
Hotspot problem
Air traffic management
Air traffic flow management
Scheduling
Job-shop scheduling
Optimization
Mixed integer programming
Linear programming

A B S T R A C T

Many regions of the world are currently struggling with congested airspace, and Europe is no exception. Motivated by our collaboration with relevant European authorities and companies in the Single European Sky ATM Research (SESAR) initiative, we investigate novel mathematical models and algorithms for supporting the Air Traffic Flow Management in Europe. In particular, we consider the problem of optimally choosing new (delayed) departure times for a set of scheduled flights to prevent en-route congestion and high workload for air traffic controllers while minimizing the total delay. This congestion is a function of the number of flights in a certain sector of the airspace, which in turn determines the workload of the air traffic controller(s) assigned to that sector. We present a MIP model that accurately captures the current definition of workload, and extend it to overcome some of the drawbacks of the current definition. The resulting scheduling problem makes use of a novel formulation, Path&Cycle, which is alternative to the classic big-*M* or time-indexed formulations. We describe a solution algorithm based on delayed variable and constraint generation to substantially speed up the computation. We conclude by showing the great potential of this approach on randomly generated, realistic instances.

1. Introduction

Air Traffic Flow Management consists in regulating air traffic to prevent congestion while optimizing the usage of the available capacity. This is a very important topic in SESAR (SESAR, 2020), an ongoing extensive European collaborative project with the objective of improving and modernizing every aspect of the air traffic management in Europe. In this paper, we present recent results related to our work on SESAR sub-projects.

The European airspace is divided into control sectors, each assigned to one or more air traffic controllers who are in charge of guaranteeing the safety of the air traffic. The tasks of a controller (e.g., communicate with pilots, handover flights to the controller of an adjacent sector, prevent potential conflicts, etc.) are heavily regulated, each one requiring a certain amount of time and effort. Within each sector, controllers are solving the Air Traffic Control problem of finding a feasible combination of flight paths, subject to regulations about the temporal and spatial separation of airplanes. In Europe, these flight paths are also subject to the schedule determined by the central control authority. The workload of a controller is usually a function of these tasks which, in turn, is a function of the flights traversing the sector. Therefore, capacity constraints are imposed on each sector to regulate the number of flights and consequently to keep the workload of controllers within the set boundaries. In Europe, this capacity is computed through involved simulation procedures, standardized by the European control authority. Briefly, depending on the traffic scenario, each task performed by a controller is decomposed into atomic controlling actions, and each such action has an expected time required for execution. For a given traffic scenario, the expected overall time use necessary for a controller in a time window of one hour must not exceed 42 min (70% workload). Then, many historical traffic

☆ This work was funded by The Research Council of Norway [grant numbers 237718, 267554].

* Corresponding author at: Department of Mathematics, P.O box 1053 Blindern, 0316 Oslo, Norway.

E-mail address: andreana@math.uio.no (A. Nakkerud).

https://doi.org/10.1016/j.cor.2020.105159
0305-0548/© 2020 The Author(s). Published by Elsevier Ltd.

scenarios are assessed and, for each such scenario, the maximum number of flights which can be controlled within the time threshold is determined. The capacity of the sector is then simply the minimum of such numbers overall traffic scenarios (Flynn et al., 2003). Whenever a set of flights violates these capacity regulations in a certain sector, we say there is a *hotspot* (Allignol et al., 2012; Dubot et al., 2016) (see Fig. 1 for an illustrated example). In this paper, we present techniques that allow more precise modelling of controller workload, which in turn could be used to allow higher capacities without the risk of overloading a sector.

Typically, the working day of a controller is divided into fixed time windows, e.g., from 10 a.m. to 11 a.m., from 11 a.m. to 12 p.m., and so on. Then the regulations usually impose a limit on the number of aircraft entering the sector during each time window. Since flight plans are submitted to the control authorities by each airline only a few hours before departure (sometimes even half an hour in the case of private jets), it is not uncommon that they will ultimately produce a hotspot. Control authorities have many ways to deal with this issue, such as asking the airline to submit an altered flight plan. When a satisfactory solution cannot be agreed upon in due time, control authorities can simply choose to delay the departure. This is the scenario considered in this paper. In particular, given the schedule for a set of flights and given the capacity constraints for all sectors, the Hotspot Problem (HP) consists in finding new (possibly delayed) departure times for all flights such that the corresponding schedule is hotspot-free and the sum of delays is minimized.

The exact definition of HP depends on the exact definition of *hotspot*. As mentioned above, the current definition agreed in Europe (Flynn et al., 2003) considers fixed intervals of time and, for each sector and for each interval of time, it compares the aircraft entry counts with the predefined sector capacity. There are several drawbacks with this definition. The most evident is perhaps a phenomenon called bunching, where the excess flights of a certain interval of time are simply moved and amassed at the beginning of the next interval. We propose to solve this by considering sliding intervals of time, where the sector capacity must be respected in any interval of time of a certain length. A recent study (Guibert et al., 2019), which also involves the European control authority, considers the possibility to jointly take into account entry counts and occupancy counts. The first measures the number of aircraft entering a sector within a certain interval of time, whereas the latter measures the number of aircraft simultaneously traversing a sector within a certain interval of time (see Fig. 2). In this paper, we describe a model that considers both sliding intervals of time and occupancy counts, and that can be immediately extended to support fixed interval of times and entry counts.

The literature on the hotspot problem is small, with only a few papers dealing with somewhat related problems (of which an overview can be found in Zhong (2018)). None of these papers matches exactly with the Hotspot Problem. In Schefers et al. (2017), the airspace is subdivided into micro-cells of unit capacity, and airplanes can be delayed at the departure, but only within the assigned time slot. The question of delaying flights to reduce congestion is discussed in de Jonge and Seljée (2011) along with a greedy-like prioritization-based iterative algorithm to compute delays. In contrast, a MILP model for the same problem is discussed in Damhuis et al. (2015), with some interesting experimental results on some simulation scenarios. In Vaaben and Larsen (2015) the problem is studied from the side of the airlines. When control authorities issue flying restrictions, airlines need to modify flight trajectories to meet such restrictions. Both schedules and trajectories can be modified in this study, but the feasible trajectories are chosen from a predefined, finite set. In Sailauov and Zhong (2016) the standpoint is again from the control authority side. The model factors in many details, including, for instance, stops at intermediate airports and fairness of the solutions. The resulting, overarching time-indexed MILP model is probably too complex to be solved for the size of practical instances, and indeed the experiments reported in the paper involves only two flights.

One of the most closely related problems in the literature is presented in Kim et al. (2009), where a combination of greedy and randomized rounding heuristics is used to minimize the maximum occupancy of any sector. Our problem mainly differs from the one in Kim et al. (2009) in that we minimize delays while respecting capacity restrictions rather than minimize occupancy, and that we apply an exact method. Our computational experiments (Section 7) are on instances of the single-sector problem, which is already NP-hard in general Kim et al. (2009). Our ability to solve these instances using our exact method rests on the fact that we have an available schedule which is already close to feasible. If no schedule is available, a heuristic method like the one presented in Kim et al. (2009) could be applied as a preprocessing step.

Most of the above-mentioned papers focus on modelling issues, using either constraint (CP) or mathematical programming—mainly Mixed Integer Programming (MIP). The resulting formulations are then solved by directly invoking a MIP solver (with the exception of Damhuis et al. (2015) and Kim et al. (2009) where delayed constraint generation is applied). In our experience, however, this approach typically does not suffice to find exact solutions to instances of a practical size. The main reason is that the classical formulations for this class of problems are either too weak or too large to work well in practice without additional algorithmic enhancements. More specifically, the Hotspot Problem is a variant of the job-shop scheduling problem with blocking and no-wait constraints (see Mascis and Pacciarelli, 2002), where the sectors can be seen as machines and the flights as jobs. The main issue of this class of problems is that we have to introduce disjunctive constraints to represent and solve conflicts in the use of shared resources. Basically, two MIP models are competing in the literature: the big-M and the time-indexed formulations (see Queyranne and Schulz, 1994). The former usually provides weak bounds, and thus large search trees; the latter produces better bounds but at the cost of increasing time to solve the relaxations. While time-indexed formulations work well for some scheduling problems, it is a well-known problem that they struggle when the number of time slots becomes large, which is often the case in modern transportation scheduling (Mannino and Mascis, 2009). So, the reduction in the number of branching nodes due to a stronger bound is typically not enough to compensate for the increase in running times.

Finally, while the Hotspot Problem takes into account en-route conflicts, it only allows delaying airplanes at the airports. The Hotspot Problem, therefore, belongs to a larger class of air traffic scheduling problems in terminal control areas, such as Avella et al. (2017), Kim et al. (2009), Bianco et al. (2006), D'Ariano et al. (2015), Samà et al. (2017).

In this paper, we experiment with a different MIP formulation for job-shop scheduling in transportation, recently introduced by Lamorgese and Mannino (2019) for rail traffic management, and then extended to cope with air traffic in Mannino and Sartor (2018). The *Path&Cycle formulation* is a MIP formulation for job-shop scheduling problems. As the classical big-M formulation, it uses a set of binary variables per disjunction, but without resorting to the notorious big-M coefficients and constraints. This allows for stronger relaxations, without the excessive increase in running times typically associated with time-indexed formulations. For the Hotspot Problem, the Path&Cycle formulation was indeed proven to be more effective than the big-M formulation (Mannino and Sartor, 2018). In this paper, we develop a Path&Cycle formulation for the Hotspot Problem, where delayed constraint generation is applied to cope with the potentially large number of constraints.

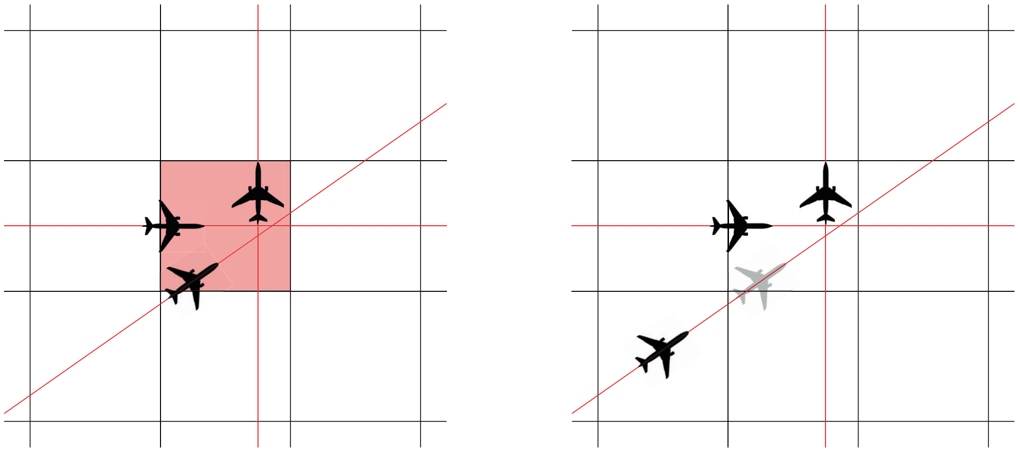


Fig. 1. In this illustration a sector is considered to be a hotspot if it is occupied by more than two flights at any point in time. On the left, the highlighted sector is a hotspot. On the right, we have delayed one flight to resolve the hotspot.

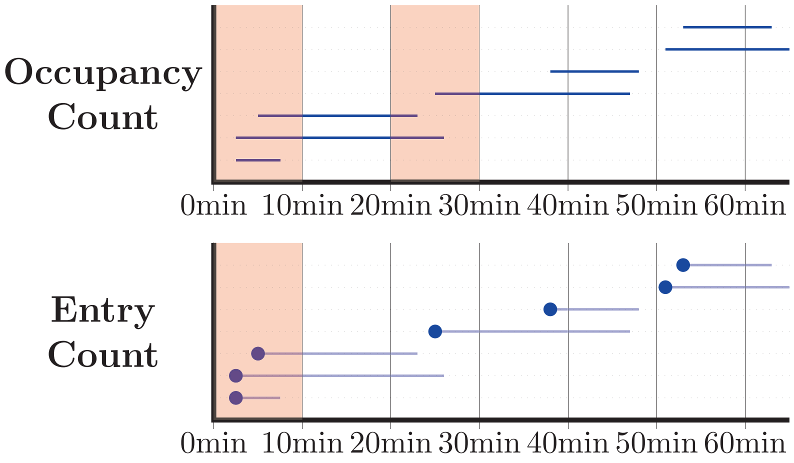


Fig. 2. Occupancy and entry counts with fixed windows of capacity 2. Aircraft entries are shown as dots to illustrate that only the first 10-minute window contains more than 2 entries.

To speed up the separation process, we also describe a projection-based, preprocessing technique that has the potential to significantly reduce the size of the *alternative graph* which is the input to our solution algorithm. This projection technique can also be applied to other problems that can be modelled using an alternative graph, for example, train scheduling. Finally, since real-life instances were not available, we tested our approach on randomly generated instances of sizes comparable with the real ones.

A preliminary to this work (Mannino et al., 2018) was presented at SESAR Innovation Days 2018, organized by the Single European Sky ATM Research (SESAR) Joint Undertaking under the Horizon 2020 framework.

Our contribution is twofold: modelling and algorithmic. From the modelling standpoint, we introduce and model the concepts of sliding window capacity constraints (Section 3.1) and and fixed window capacity constraints (Section 5.1), counting either occupancy (Section 3.2 and Mannino and Sartor (2018)) or entries (Section 5.2) within the specific window towards capacity. Finally, we discuss layering of capacity constraints (Section 5.3). These extensions allow for a better representation of the actual controller workload with respect to previous works. From the algorithmic standpoint, we have extended the Path&Cycle solution algorithm to tackle these model extensions.

The rest of the paper is organized as follows. In Section 2, we describe in detail the Hotspot Problem, and in Section 3 we introduce a mathematical model for the Hotspot Problem. We introduce the Path&Cycle model for the Hotspot Problem in Section 4, and in Section 5 we introduce extensions to this model. We discuss our solution algorithm and its implementation in Section 6, and our computational results in Section 7.

2. The hotspot problem

We are given a set $\mathscr{F}$ of flights through an airspace divided into predefined control sectors $\mathscr{S}$. Conventionally, we also include in $\mathscr{S}$ a fictitious "arrival sector" a_f for every $f \in \mathscr{F}$, to represent the final sector of flight *f*. Note that a_f may be an airport or the space outside the control region. Also, we conventionally assume $a_f \neq a_g$ for any pair of distinct flights $f, g \in \mathscr{F}$. The route of each flight $f \in \mathscr{F}$ is given as an ordered set $\mathscr{S}_f = \{s_1^f, s_2^f, \dots, s_{n_f}^f\}$ of sectors, with $a_f = s_{n_f}^f$. Also, we let $d_f = s_1^f$ be the departure sector (departure sectors of different flights may coincide). Note that the first sector s_1^f may be the one immediately after the flight takes off from a controlled airport, or the first sector in the controlled airspace when the flight enters the control space already airborne. The *travel time* Λ_f^s of flight *f* through sector $s \in \mathscr{S}_f$ is the time *f* uses to travel through *s* (we let $\Lambda_f^{a_f} = 0$). The *release time* Γ_f of *f* is either (i) the (expected) earliest departure time of *f* (if the departure airport is within the controlled airspace), or (ii) the time *f* enters the controlled airspace. Release times are given relative to an arbitrary *reference time* t_o. Because the task is to schedule flights sometime in the future, normally (but not necessarily) t_o coincides with the time when such planning is carried out. We may assume t_o precedes all release times, so we have $\Gamma_f \geqslant 0$ for all *f*.

The planning horizon *H* is subdivided into a set of contiguous intervals, the *time windows*. The subdivision is completely defined by the starting time and the size of each time window. Typically, one-hour intervals are considered, starting at 12 a.m., 1 a.m., …, but other window sizes may be possible. Each control sector $s \in \mathscr{S}$ is assigned a capacity $c_s \in \mathbb{Z}$. There are two alternative ways to interpret such capacity.

2.1. Entry count

In this interpretation, the capacity c_s of a sector represents the maximum number of flights which can enter the sector in the time window. Indeed, if things go according to plan, most of the (time-consuming) activities carried out by controllers occur when a flight enters the sector. This is the definition used by the European control authorities.

2.2. Occupancy count

The capacity c_s of a sector represents the maximum number of flights which can be in the sector during the time window. Note that, depending on the travel time and window size, entry and occupancy counts may differ significantly (see Fig. 2).

We finally define the schedule of a flight $f \in \mathscr{F}$ as the time $t_s^f \in \mathrm{IR}$ the flight enters sector $s \in \mathscr{S}_f$.

The Hotspot Problem is then the problem of finding a feasible schedule $\mathbf{t}^f$ for each flight such that, for each sector $s \in \mathscr{S}$, the number of flights in *s* never exceeds the capacity c_s of the sector in any predefined time window.

Note that since the travel time in each sector is fixed, that is, it is subject to a no-wait constraint, the schedule of a flight is completely determined once the departure time is established. As a consequence, flights entering the controlled space from "outside" will have fixed release times, and their schedules cannot be modified.

2.3. Fixed and sliding windows

The official definition of hotspot (Flynn et al., 2003) is based on the above described *fixed window* approach. This approach has some major drawbacks. The first is that it may lead to unwanted violations of capacity. Suppose we ensure that in given fixed windows, say from 10 a.m. to 11 a.m. and from 11 a.m. to 12 a.m., the number of aircraft in a given sector does not exceed the capacity, say 10. This does not prevent that in "intermediate" windows, say from 10:30 a.m. to 11:30 a.m., we have up to 20 flights in the sector. The second is mainly an operational problem, namely *bunching*. When the capacity of a sector is violated during a time window, one possibility is to hold some of the involved flights at the airport. Obviously, one would like to generate the least possible delay, so a natural algorithm is to hold the flights precisely the time necessary to skip the overloaded time window. However, this may produce a schedule with many flights bunching up at the beginning of the next time window.

An alternative approach that solves both the "intermediate" capacity violations and bunching is the *sliding window*. It simply states that for *any* time interval of a given size, the number of flights in a sector should not exceed the capacity of the sector. In other words, the starting time of the control window is not fixed and can be anywhere in the time horizon. Depending on the adopted time window, we have different definitions of a hotspot.

2.4. Fixed window hotspot

A *fixed window* hotspot occurs when too many flights occupy the sector during one of the predefined fixed windows. In Fig. 3 the windows have width 10 min, and start at time $0, 10, 20, \dots$. In this example, the windows do not overlap, but in principle, they could. The red shaded areas of the fixed window portion of Fig. 3 shows when the sector is a fixed window hotspot.

2.5. Sliding window hotspot

A *sliding window* hotspot occurs when too many flights occupy the sector during any period of time at most as long as the window width Δ. It may be viewed as a family of overlapping fixed windows, where the next window starts one unit of time after the previous. In Fig. 3, the sliding window has width 10 min. The red shaded areas of the sliding window portion of Fig. 3 shows when the sector is a sliding window hotspot. The grey crosshatched area around the last hotspot shows how far that window can slide with the sector remaining a hotspot.

We summarize the above with the following formal definition.

Definition 1. Let $n_s(t)$ be some count of flights in sector *s* at time *t* based on the current planned departure times. The sector *s* is a *hotspot* at time *t* if $n_s(t) > c_s$, where c_s is the capacity of *s*.

The form of $n_s(t)$ in the above definition depends on the type of the corresponding capacity constraint. We will use Δ to denote window width, and we will subscript Δ to denote widths of specific windows. We let $W(t)$ denote the window corresponding to time *t*, that is either $W(t) = W_{\text{fixed}}(t) = [a, a + \Delta]$ is the fixed window interval such that $t \in [a, a + \Delta]$, or $W(t) = W_{\text{sliding}}(t) = [t, t + \Delta]$ is the sliding window starting at *t*. Let $I_f^s = [t_f^0, t_f^1]$ be the time interval flight *f* spends in sector *s*. Then

$$n_s(t) = \begin{cases} |\{f \in \mathscr{F} : I_f^s \cap W(t) \neq \emptyset\}|, & \text{for occupancy counts} \\ |\{f \in \mathscr{F} : t_f^0 \in W(t)\}|, & \text{for entry counts.} \end{cases} \quad (1)$$

In order to model controller workload more realistically, we also propose layering different hotspot definitions. We propose using a long-term window to manage sustained workload, and a simultaneous short-term window to manage peak workload (Section 5.3). In this case, the capacity may not depend only on the sector, but also on the specific capacity constraint.

3. Modelling the sliding window hotspot problem

In this section, we present our MILP model for the sliding window hotspot problem. This model is an extension of the novel Path&Cycle formulation, which is a special version of the job-

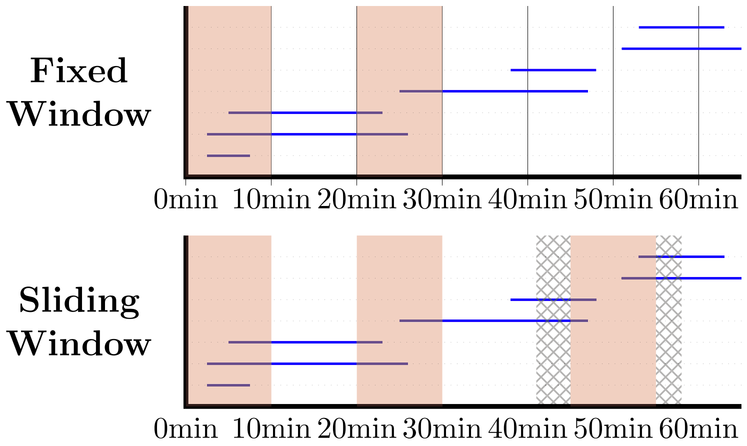


Fig. 3. Fixed time windows versus sliding time windows. The red boxes (and line) illustrate times when the section becomes a hotspot. In each case, we have set the capacity to 2. In the sliding window case, the windows can move slightly to the left and right with the sector remaining a hotspot, as illustrated by the grey crosshatched areas on the right side of the figure. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

shop scheduling problem with blocking and no-wait constraints (Mascis and Pacciarelli, 2002). The formulation was recently introduced by Lamorgese and Mannino in Lamorgese and Mannino (2019) for train scheduling and then extended by Mannino and Sartor in Mannino and Sartor (2018) to flight scheduling.

Let $f \in \mathscr{F}$ be a flight and $s \in \mathscr{S}_f$ be a sector on the flight plan $\mathscr{S}_f = \{d_f = s^f_1, s^f_2, \dots, s^f_{n_f} = a_f\}$ of f. With some abuse of notation, if $s \in \mathscr{S}_f \setminus \{s^f_{n_f}\}$, we denote by t^{s+1}_f the time f enters the sector following s in the flight plan.

Using this notation, since t^{s+1}_f is also the time when f leaves sector s. We get the *schedule constraints*

$$t^{s+1}_f - t^s_f = \Lambda^s_f \tag{2}$$

that is the travel time of f through s is precisely Λ^s_f.

Next, recalling that the release time $\Gamma_f \geqslant 0$ is given relative to the *reference time* t_o, we have

$$t^{d_f}_f - t_o \geqslant \Gamma_f \tag{3}$$

for all flight f, with equality holding if f enters the controlled region already airborne.

Any feasible schedule $\mathbf{t}$ must satisfy (2) and (3). Linear constraints of the form $t_v - t_u \geqslant l_{uv}$ are called *time precedence* constraints. We note that (2) can be written as a pair of time precedence constraints.

3.1. Modelling sliding window constraints

Let Δ be the size of the sliding window and $\epsilon > 0$ be a small constant. We say that two distinct flights $f, g \in \mathscr{F}$ *do not meet* in a sector $s \in \mathscr{S}_f \cap \mathscr{S}_g$ if either (i) f leaves sector s at least Δ units of time before g enters it, that is $t^s_g - t^{s+1}_f \geqslant \Delta + \epsilon$; or (ii) g leaves sector s at least Δ units of time before f enters it, that is $t^s_f - t^{s+1}_g \geqslant \Delta + \epsilon$. In contrast, we say that f, g *meet* in s if none of the two conditions (i) and (ii) is satisfied, that is,

$$t^s_g - t^{s+1}_f \leqslant \Delta \ \wedge \ t^s_f - t^{s+1}_g \leqslant \Delta.$$

Because f, g either meet or do not meet in sector s, we have that any feasible schedule must satisfy the following constraint:

$$t^s_g - t^{s+1}_f \geqslant \Delta + \epsilon \ \bigvee \ t^s_f - t^{s+1}_g \geqslant \Delta + \epsilon \ \bigvee \ t^s_g - t^{s+1}_f \leqslant \Delta \ \wedge \ t^s_f - t^{s+1}_g \leqslant \Delta. \tag{4}$$

The above constraint is called a *disjunctive constraint*, and it is the disjunction of three terms. The first two terms are time precedence constraints, whereas the third term is a conjunction of two time-precedence constraints. Any feasible schedule $\mathbf{t}$ will satisfy exactly one of the three terms in disjunction (4).

In order to represent this disjunctive constraint in a MILP, we introduce the *selection variables*. For any ordered pair (f, g) of flights f, g both flying through a common sector $s \in \mathscr{S}_f \cap S_g$, we define the *precedence variable*

$$y^s_{fg} = \begin{cases} 1, & \text{if } f \text{ precedes } g \text{ in } s \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

and for any unordered pair $\{f, g\}$ of distinct flights f, g both flying through a common sector s, we define the *meeting variable*

$$z^s_{fg} = \begin{cases} 1, & \text{if } f \text{ and } g \textit{ mhboxmeetin } s \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where we have preferred the notation z^s_{fg} for $z^s_{\{fg\}}$, with the convention that z^s_{fg} is the same as z^s_{gf}.

So, associated with a pair of distinct flights f, g with a shared sector s, we have three binary variables: two precedence variables and one meeting variable. Each of these variables corresponds to one of the terms of the disjunction (4) associated with f, g, and s. Since exactly one of the three terms must be satisfied by any feasible schedule, the binary variables must satisfy the following *selection constraints*:

$$y^s_{fg} + y^s_{gf} + z^s_{fg} = 1. \tag{7}$$

We use $(\mathbf{y}, \mathbf{z})$ to denote the vector of selection variables, that is, the *selection vector*. A schedule $\bar{t}$ associated with a selection vector $(\bar{y}, \bar{z})$ must satisfy, for all pair of distinct flights $f, g \in \mathscr{F}$, and all $s \in \mathscr{S}_f \cap \mathscr{S}_g$

$$\begin{array}{llll} & (i) & t^s_g - t^{s+1}_f \geqslant \Delta + \epsilon, & \text{if } \ \bar{y}^s_{fg} = 1 \\ \text{or} & (ii) & t^s_f - t^{s+1}_g \geqslant \Delta + \epsilon, & \text{if } \ \bar{y}^s_{gf} = 1 \\ \text{or} & (iii) & t^s_g - t^{s+1}_f \leqslant \Delta \ \wedge \ t^s_f - t^{s+1}_g \leqslant \Delta, & \text{if } \ \bar{z}^s_{fg} = 1. \end{array} \tag{8}$$

In other words, the selection variables decide which time precedence constraints in each disjunction must be satisfied by the schedule $\mathbf{t}$. Associated with any selection vector (y, z) we have thus a system of precedence constraints. This motivates the next:

Definition 2. We let $A(y, z)$ be the system of precedence constraints (2) and (3), and the constraints of the disjunctions (8) associated with the selection vector (y, z).

The system of precedence constraints $A(y, z)$ ensures that each flight's schedule is feasible in isolation and that the precedence decisions coded by $\mathbf{y}$ and $\mathbf{z}$ are respected. However, this system of constraints does not take into account capacity.

3.2. Capacity constraints

The vector $\mathbf{z}$ determines which pairs of flights meet in a given sector. We say that a set of flights $F \subseteq \mathscr{F}$ *meet* is a shared sector s if there is a time window of size Δ when all flights are in s. Now, a set $F \subseteq \mathscr{F}$ of flights meet in a sector s if and only if every pair $f, g \in F$ of distinct flights meet in s. In this case we have that $z^s_{fg} = 1$ for all distinct $\{f, g\} \subseteq F$ and $\sum_{\{f,g\} \subseteq F} z^s_{fg} = \binom{|F|}{2}$.

We denote by $\mathscr{F}_s \subseteq \mathscr{F}$ the set of flights going through s and let c_s be the capacity of s, i.e. no set of $c_s + 1$ flights can meet in s. Then, for every set of flight $F \subseteq \mathscr{F}_s$ where $|F| = c_s + 1$, $\mathbf{z}$ must satisfy the following constraint:

$$\sum_{\{f,g\} \subseteq F} z^s_{fg} \leqslant \binom{c_s + 1}{2} - 1. \tag{9}$$

A meeting (selection) vector is *hotspot free* if it satisfies (9).

3.3. A disjunctive formulation for the Hotspot problem

We can now state more formally our basic version of the Hotspot problem.

3.3.1. The sliding window hotspot problem

The Sliding Window Hotspot problem amounts to finding a hotspot free selection vector $(\mathbf{y}, \mathbf{z})$ satisfying (7) and (9), and a schedule $\mathbf{t}$ satisfying $A(y, z)$, such that the cost $w(\mathbf{t})$ of the schedule is minimized.

The function $w : \mathbf{t} \to \mathbb{R}$ determines the cost of the schedule and typically increases monotonically with the delays (that is with $\mathbf{t}$)). Note that constraints (2), (3), (8) and (9), provide a *disjunctive formulation* for the feasible solutions $(\mathbf{y}, \mathbf{z}, \mathbf{t})$ to the Hotspot problem. A standard way to linearize (8) is by means of the so called big-M trick (see, for instance, Queyranne and Schulz, 1994). However, as discussed in the introduction, this leads to very weak relaxations. Following (Lamorgese and Mannino, 2019), we prefer a different model.

4. The Path&Cycle model

4.1. The route node graph

At the heart of the Path&Cycle formulation is an event graph $G = (V, E)$ called the *route node graph*. The nodes are associated with the time variables of the disjunctive formulation for the hotspot problem. In particular, V contains a special node o, the *origin*, associated with the reference time variable t_o and a *route node* $\langle f, s\rangle$ associated with the variable t^s_f, for each $f \in \mathscr{F}, s \in \mathscr{S}_f$. The node $\langle f, s\rangle$ represents the event "flight f enters sector s". The directed arcs of the graph are associated with time precedence constraints between the schedule variables. So, if $u = \langle f, s\rangle \in V$ and $v = \langle g, r\rangle \in V$, then $(u, v) \in E$ with length l_{uv} represents the constraint $t^g_r - t^f_s \geqslant l_{uv}$. Note that the inequality $t_u - t_v \leqslant l$ is equivalent to $t_v - t_u \geqslant -l$, which in turn is associated with an arc (u, v) with length $-l$. Furthermore, an equality precedence constraint $t_v - t_u = l$ can be transformed into two inequalities, which in turn correspond to two anti-parallel arcs (u, v) and (v, u) of length l and $-l$, respectively.

Release time constraints (3) are thus represented by arcs from the origin to the node associated with the first sector of every flight. Fig. 4 shows an example of a route node graph for two flights.

When two flights f and g both fly through a common sector s, then either they meet in s or one precedes the other as represented by (4). Since each term in the disjunction is either a time precedence constraint or the conjunction of two time-precedence constraints, it can be represented in the route node graph by arcs called *alternative arcs*, as shown in Fig. 5. The alternative arcs, which are drawn in the figure as either dashed or dotted arrows, have length equal Δ or $-\Delta$, according to (8). That is, the arc lengths are determined by the window width Δ. Note that we have two arcs associated with the "meet" case of the disjunction. Now, each alternative arc (or pair of arcs) is associated with one term in the disjunctive constraint (4), which in turn is associated with a selection variable. Consequently, each alternative arc is associated with exactly one selection variable. For instance, in Fig. 5, the arc $(\langle g, s_2\rangle, \langle f, s_3\rangle)$ is associated with the meeting variable $z^{s_2}_{fg}$.

We now summarize the above construction by giving a formal definition of the route graph.

Definition 3. Let $\mathscr{F}$ be a set of flights and, for $f \in \mathscr{F}$, let Γ_f be the departure time, and let Λ^s_f be the travel time for each $s \in \mathscr{S}_f$. Let $\mathscr{F}^* \subseteq \mathscr{F}$ be the set of flights with fixed departure times. Let $\mathscr{U}_{\mathscr{F}} = \{\langle f, s\rangle : f \in \mathscr{F}, s \in \mathscr{S}_f\}$ be the set of route nodes. The *route node graph* is a directed graph $G = (V, E_R \cup E_T \cup E_A)$, where E_R are the *release arcs*, E_T are the *fixed precedence arcs*, and $E_A = E^y_A \cup E^z_A$ are the *alternative arcs*. We have

$$\begin{aligned}
V &= \mathscr{U}_{\mathscr{F}} \cup \{o\}\\
E_R &= \{(o, \langle f, d_f\rangle) : f \in \mathscr{F}\}\\
&\quad \cup \{(\langle f, d_f\rangle, o) : f \in \mathscr{F}^*\}\\
E_T &= \{(\langle f, s\rangle, \langle f, s+1\rangle) : f \in \mathscr{F}, s \in \mathscr{S}_f\}\\
&\quad \cup \{(\langle f, s+1\rangle, \langle f, s\rangle) : f \in \mathscr{F}, s \in \mathscr{S}_f\}\\
E^y_A &= \{(\langle f, s+1\rangle, \langle g, s\rangle) : s \in \mathscr{S}, (f, g) \in \mathscr{F}_s \times \mathscr{F}_s, f \neq g\}\\
E^z_A &= \{(\langle f, s\rangle, \langle g, s+1\rangle) : s \in \mathscr{S}, (f, g) \in \mathscr{F}_s \times \mathscr{F}_s, f \neq g\}.
\end{aligned}$$

For each arc $e \in E$, its length l_e is the constant term in the corresponding time precedence constraints. So, for $e = (o, \langle f, d_f\rangle) \in E_R$ we have $l_e = \Gamma_f$; for $e = (\langle f, d_f\rangle, o) \in E_R$ we have $l_e = -\Gamma_f$; for $e = \{(\langle f, s\rangle, \langle f, s+1\rangle) \in E_T$ we have $l_e = \Lambda_s$; for $e = \{(\langle f, s+1\rangle, \langle f, s\rangle) \in E_T$ we have $l_e = -\Lambda_s$; for $e \in E^y_A$ with have $l_e = \Delta$; and finally, for $e \in E^z_A$ we have $l_e = -\Delta$. If e is an alternative arc, we let $\mathrm{Var}(e)$ be the variable associated with e. Since every alternative arc is associated with a selection variable in the binary vector $(\mathbf{y}, \mathbf{z})$, we may interpret $(\mathbf{y}, \mathbf{z})$ as the incidence vector of a subset $E_A(\mathbf{y}, \mathbf{z}) \subseteq E_A$. We use $G(\mathbf{y}, \mathbf{z}) = (V, E(\mathbf{y}, \mathbf{z}))$ to denote the subgraph of G induced by the set of arcs $E(\mathbf{y}, \mathbf{z}) = E_R \cup E_T \cup E_A(\mathbf{y}, \mathbf{z})$, that is, the graph we obtain from G by removing all alternative arcs not selected in $(\mathbf{y}, \mathbf{z})$.

We extend the definition of Var to a set of arcs $E = \{e_1, \ldots, e_n\}$ so that $\mathrm{Var}(E) = \{\mathrm{Var}(e_i) : e_i \in E\}$.

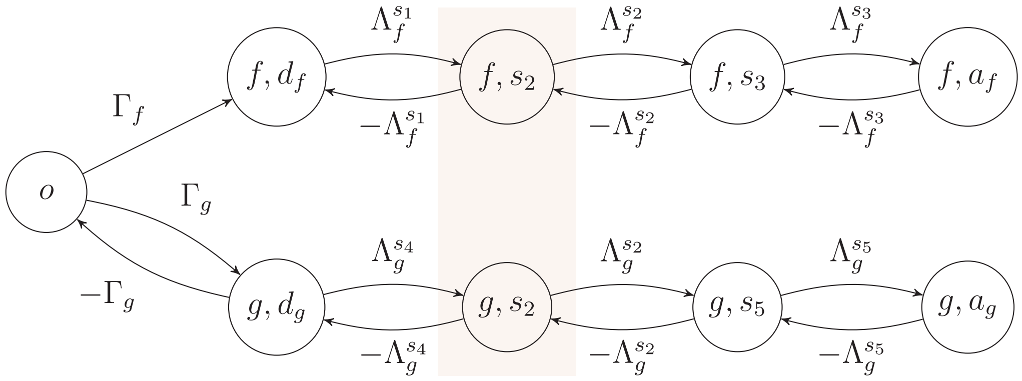


Fig. 4. Route node graph for two flights f and g. g has a fixed departure time $t_o + \Gamma_g$, while f has an earliest departure time $t_o + \Gamma_f$. Both flights fly through sector s_2. The arcs represent time precedence constraints.

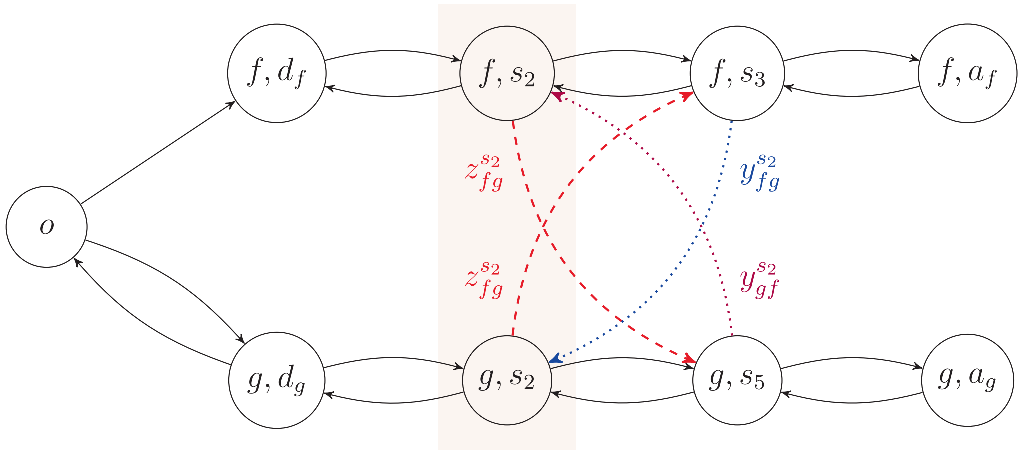


Fig. 5. Since both flights f and g fly through s_2, they must either meet there, or one must precede the other. The dashed and dotted *alternative arcs* between nodes of different flights represent each of these options. All the arc lengths have been omitted, and the alternative arcs have instead been labelled with their associated binary variables.

4.2. Positive cycles and longest paths

By construction, the route node graph contains a directed path from the origin o to any other node of the graph. Indeed, there is a directed edge from o to the node $\langle f, d_f \rangle$ associated with the first sector in the route of flight $f \in \mathscr{F}$, and a directed path from $\langle f, d_f \rangle$ to every node $\langle f, s_f \rangle$ associated with any sector $s_f \in \mathscr{S}_f$ on the route of f. Note that this path does not make use of alternative edges, and so for any selection vector $(\mathbf{y}, \mathbf{z})$, the graph $G(\mathbf{y}, \mathbf{z}) = (V, E(\mathbf{y}, \mathbf{z}))$ also contains a path P_u from o to any other node $u \in V$. By construction, even if G contains negative length arcs, the length $l(P_u)$ of this path is always non-negative.

It is well known that, if $G(\mathbf{y}, \mathbf{z})$ does not contain a strictly positive directed cycle (corresponding to an infeasible subset of time precedence constraints), then it contains a maximum length path from o to any other node u. For all $u \in V$, let us denote by $L^*(\mathbf{y}, \mathbf{z}, u)$ such length. The following Lemma 4 follows from well-known results (see, for instance, Bertsimas and Tsitsiklis, 1997).

Lemma 4. *Let $(\bar{y}, \bar{z})$ be a selection vector. There exists a feasible solution to the set of inequalities $A(\bar{y}, \bar{z})$ if and only if the graph $G(\bar{y}, \bar{z})$ does not contain strictly positive length directed cycles. Then, a feasible solution is given by $t_u^* = L^*(\bar{y}, \bar{z}, u)$, for $u \in V$. Finally, if the cost function $w(\mathbf{t})$ is non-decreasing, then $\mathbf{t}^*$ is a feasible solution which minimizes $w(\mathbf{t})$.*

We let $\mathscr{C}^+$ be the set of all strictly positive directed cycles of the route node graph G and let $C \in \mathscr{C}^+$ be one such cycle. We denote by $\mathrm{Alt}(C) = C \cap E_a$ the set of alternative arcs in C. We divide the alternative arcs into precedence arcs $\mathrm{Alt}_{\mathbf{y}}(C) = C \cap E_a^y$, and meeting arcs $\mathrm{Alt}_{\mathbf{z}}(C) = C \cap E_a^z$. It follows from Lemma 4 that, for any feasible solution $(\mathbf{y}, \mathbf{z}, \mathbf{t})$ to the hotspot problem, the graph $G(\mathbf{y}, \mathbf{z})$ does not contain a strictly positive directed cycle. So, for any $C \in \mathscr{C}^+$, at least one alternative arc in C must be excluded in any feasible solution, and the selection vector $(\mathbf{y}, \mathbf{z})$ satisfies the following set of (no good) constraints:

$$\sum_{e \in \mathrm{Alt}_{\mathbf{y}}(C)} y_e + \sum_{e \in \mathrm{Alt}_{\mathbf{z}}(C)} z_e \leqslant |\mathrm{Alt}(C)| - 1, \qquad C \in \mathscr{C}^+. \tag{10}$$

4.3. Path&Cycle MILP formulation

To simplify the following discussion, we assume now that the objective function amounts to minimizing the sum of the delays of all flights at their destinations; the extension to any function non-decreasing with time is straightforward. With this assumption, it follows by Lemma 4 that the Hotspot problem can be modelled as follows. We let $\mathscr{A} = \{\langle f, a_f\rangle : f \in \mathscr{F}\}$ be the set of arrival route nodes, and get the mathematical model

$$\begin{array}{lll} \min & \sum_{u\in\mathscr{A}} L^*(\mathbf{y},\mathbf{z},u) & \\ \text{s.t.} & & \\ (i) & y^s_{fg} + y^s_{gf} + z^s_{fg} = 1 & s\in\mathscr{S}, \{f,g\}\subseteq\mathscr{F}_s \\ (ii) & \sum_{e\in \mathrm{Alt}_{\mathbf{y}}(C)} y_e + \sum_{e\in \mathrm{Alt}_{\mathbf{z}}(C)} z_e \leqslant |\mathrm{Alt}(C)| - 1 & C\in\mathscr{C}^+ \\ (iii) & \sum_{\{f,g\}\subseteq F} z^s_{fg} \leqslant \binom{|F|}{2} - 1 & s\in\mathscr{S}, F\subseteq\mathscr{F}_s, |F| = c_s+1 \\ & y^s_{fg}, y^s_{gf}, z^s_{fg} \in \{0,1\} & s\in\mathscr{S}, \{f,g\}\subseteq\mathscr{F}_s. \end{array} \tag{11}$$

Note that this is a non-compact formulation since the size of $\mathscr{C}^+$ in (ii) is potentially exponential in G. To turn (11) into a MILP, we need to deal with the objective function.

To this end, for an arrival node $u \in \mathscr{A}$, we introduce a non-negative, continuous variable $\eta_u \in \mathrm{IR}_+$ such that, for any $(\mathbf{y},\mathbf{z})$, we have $\eta_u \geqslant L^*(\mathbf{y},\mathbf{z},u)$. Equivalently, we want $\eta_u \geqslant l(P)$ for any path P from o to u in $G(\mathbf{y},\mathbf{z})$.

Now, let $\mathscr{P}_u$ be the set of all *simple* paths from o to u in $G = (V,E)$, let $P \in \mathscr{P}_u$, let $l(P)$ be its length, and let $\mathrm{Alt}(P)$ be the set of alternative arcs of P. If all such arcs are selected in a solution $(\mathbf{y},\mathbf{z})$ then P belongs to $G(\mathbf{y},\mathbf{z})$ and $\eta_u \geqslant l(P)$. This can be expressed by the following linear expression:

$$\eta_u \geqslant l(P)\left(\sum_{e\in \mathrm{Alt}_{\mathbf{y}}(P)} y_e + \sum_{e\in \mathrm{Alt}_{\mathbf{z}}(P)} z_e - |\mathrm{Alt}(P)| + 1\right) \tag{12}$$

where Alt is defined in the same way as for cycles. Indeed, if all selection variables in (12) are 1, then the r.h.s reduces to $l(P)$, otherwise the constraint is redundant.

Combining (11) and (12) we get the following MILP formulation

$$\begin{array}{lll} \min & \sum_{u\in A} \eta_u & \\ \text{s.t.} & & \\ (i) & y^s_{fg} + y^s_{gf} + z^s_{fg} = 1 & s\in\mathscr{S}, \{f,g\}\subseteq\mathscr{F}_s \\ (ii) & \sum_{e\in \mathrm{Alt}_{\mathbf{y}}(C)} y_e + \sum_{e\in \mathrm{Alt}_{\mathbf{z}}(C)} z_e \leqslant |\mathrm{Alt}(C)| - 1 & C\in\mathscr{C}^+ \\ (iii) & \sum_{\{f,g\}\subseteq F} z^s_{fg} \leqslant \binom{|F|}{2} - 1 & s\in\mathscr{S}, F\subseteq\mathscr{F}_s, |F| = c_s+1 \\ (iv) & \eta_u \geqslant l(P)\left(\sum_{e\in \mathrm{Alt}_{\mathbf{y}}(P)} y_e + \sum_{e\in \mathrm{Alt}_{\mathbf{z}}(P)} z_e - |\mathrm{Alt}(P)| + 1\right) & u\in A, P\in\mathscr{P}_u \\ & y^s_{fg}, y^s_{gf}, z^s_{fg} \in \{0,1\} & s\in\mathscr{S}, \{f,g\}\subseteq\mathscr{F}_s \\ & \eta_u \geqslant 0 & u\in A. \end{array} \tag{13}$$

Now, since $\eta_u \geqslant L^*(\mathbf{y},\mathbf{z},u)$ for $u \in A$, the optimal value of (13) is an upper bound on the optimal value of (11). Actually, one can show that the two formulations are equivalent, (see Lamorgese and Mannino, 2019), and the following result holds

Lemma 5. *Let $\eta^*, \mathbf{y}^*, \mathbf{z}^*$ be an optimal solution to (13) and let $t^*_u = L^*(\mathbf{y}^*,\mathbf{z}^*,u)$ for $u \in V$. Then $\mathbf{t}^*$ is an optimal solution for the Hotspot Problem, and $w(\mathbf{t}^*) = \sum_{u\in A}\eta^*_u$.*

In the next section, we show how to adapt this basic formulation to fixed windows and entry counts, respectively.

5. Model extensions

The mathematical program so far introduced is based on the sliding windows model. We now show how to extend it to fixed windows.

5.1. Modelling fixed windows

In the fixed-window hotspot problem, for each sector $s \in \mathscr{S}$ the planning time horizon is subdivided into windows $\mathscr{W}_s$. For each $w \in \mathscr{W}_s$ we let Δ_w be the size of window w (in the current air traffic control practice, Δ_w is 60 min or 25 min) and T_w be the start time. The capacity constraint now requires that, in any $w \in \mathscr{W}_s$, the number of flights is bounded by c_w. We model this by introducing new selection variables to represent the fact that a flight traverses a sector before, after, or during a given time window. We introduce variables similar to (5) and (6), namely

$$\begin{aligned} y^s_{fw} &= \begin{cases} 1, & \text{if } f \text{ flies through } s \text{ before } w \\ 0, & \text{otherwise} \end{cases} \\ y^s_{wf} &= \begin{cases} 1, & \text{if } f \text{ flies through } s \text{ after } w \\ 0, & \text{otherwise} \end{cases} \\ z^s_{fw} &= \begin{cases} 1, & \text{if } f \text{ flies through } s \text{ during } w \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{14}$$

and associated terms in the corresponding disjunctive constraint

$$\begin{array}{llll} & (i) & t^{s+1}_f \leqslant T_w - \epsilon & \text{if} \quad y^s_{fw} = 1 \\ \text{or} & (ii) & t^s_f \geqslant T_w + \Delta_w + \epsilon & \text{if} \quad y^s_{wf} = 1 \\ \text{or} & (iii) & t^{s+1}_f \geqslant T_w \ \wedge \ t^s_f \leqslant T_w + \Delta_w & \text{if} \quad z^s_{fw} = 1 \end{array} \tag{15}$$

where $\epsilon > 0$ is a suitably small constant. Note that each term in the above disjunction is a standard time precedence constraint and can be represented by an arc with suitable length in the route node graph. Fig. 6 (corresponding to Fig. 5 for the sliding window capacity constraint) shows how we represent (15) in the route node graph.

The capacity constraint for the fixed windows can now be written as follows:

$$\sum_{f\in\mathscr{F}_s} z^s_{fw} \leqslant c_w, \quad s\in\mathscr{S}, \ w\in\mathscr{W}_s. \tag{16}$$

Note that, for a given sector s, we may have fixed windows of different sizes and overlapping windows. Also, fixed window and sliding window constraints can easily be used together in a combined formulation.

5.2. Modelling entry counts

It is very easy to amend our model to count entry rather than occupancy. Without going into details, this is obtained by simply replacing, in the disjunctive constraints (4), (8) and (15), variables t^{s+1}_f and t^{s+1}_g with t^s_f and t^s_g, respectively, leaving sign and constants unchanged. Accordingly, the alternative arcs in the node-route graph will be "re-directed", so that all tails and heads will be either $\langle f, s\rangle$ or $\langle g, s\rangle$, but the lengths will stay unchanged.

The result of this is that we replace the occupancy time interval of each flight with the single moment in time when the flight enters the sector. That is, from the point of view of the algorithm, the flight exists only at the moment it enters the sector. It is therefore irrelevant when the flight leaves the sector, or, equivalently, when it enters the next sector.

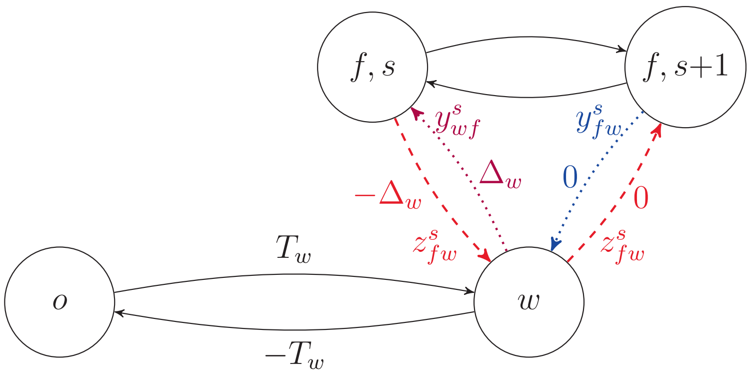


Fig. 6. Route node and arcs for a fixed window. The solid arcs between o and w represent the starting time of the window. The dashed and dotted arcs to and from w are disjunctive arcs. The disjunctive arcs are labelled with both their weights and their associated binary variables.

5.3. Modelling layered workload constraints

We are now ready to state our general MILP. The alternative arcs of the route node graph $G = (V, E)$ will be associated with each of the terms in the disjunctions (8) and (15). Then $\mathscr{C}^+$ will be the set of strictly positive directed cycles of G, whereas, for $u \in V, \mathscr{P}_u$ is the set of directed simple paths from o to u in G.

$$\begin{array}{lll}
\min & \sum_{u\in A}\eta_u & \\
\text{s.t.} & & \\
(ia) & y^s_{fg} + y^s_{gf} + z^s_{fg} = 1 & s\in\mathscr{S}, \{f,g\}\subseteq\mathscr{F}_s \\
(ib) & y^s_{fw} + y^s_{wf} + z^s_{fw} = 1 & s\in\mathscr{S}, w\in\mathscr{W}_s, f\in\mathscr{F}_s \\
(iia) & \sum_{\{f,g\}\subseteq F} z^s_{fg} \leqslant \binom{|F|}{2} - 1 & s\in\mathscr{S}, F\subseteq\mathscr{F}_s, |F| = c_s + 1 \\
(iib) & \sum_{f\in F_s} z^s_{fw} \leqslant c_w & s\in\mathscr{S}, w\in\mathscr{W}_s \\
(iii) & \sum_{e\in \mathrm{Alt}_{\mathbf{y}}(C)} y_e + \sum_{e\in\mathrm{Alt}_{\mathbf{z}}(C)} z_e \leqslant |\mathrm{Alt}(C)| - 1 & C\in\mathscr{C}^+ \\
(iv) & \eta_u \geqslant l(P)\left(\sum_{e\in\mathrm{Alt}_{\mathbf{y}}(P)} y_e + \sum_{e\in\mathrm{Alt}_{\mathbf{z}}(P)} z_e - |\mathrm{Alt}(P)| + 1\right) & u\in A, P\in\mathscr{P}_u \\
 & y^s_{fg}, y^s_{gf}, z^s_{fg} \in \{0,1\} & s\in\mathscr{S}, \{f,g\}\subseteq\mathscr{F}_s \\
 & \eta_u \geqslant 0 & u\in A.
\end{array} \qquad \text{(HP)}$$

In order to model multiple simultaneous fixed windows, we simply add more windows to the sets $\mathscr{W}_s$. We can also model multiple simultaneous sliding windows, but then we must add indexed copies of the selection variables (5) and (6) with corresponding copies of (ia) and (iia).*Summary of notation* The schedule related variables η_u are introduced in Section 4.3, while the decision variables $y^s_{fg}, z^s_{fg}, y_{fw}, y_{wf}$, and z_{fw} are introduced in Eqs. (5), (6) and (14). $\mathscr{S}$ is the set of all sectors, $\mathscr{F}_s$ is the set of all flights using sector s, and $\mathscr{W}_s$ is the set of all fixed windows of sector s. c_s is the sliding window capacity of sector s, and c_w is the capacity of the fixed window w. The sets $\mathscr{C}^+$ (of strictly positive cycles) and $\mathscr{P}_u$ (of longest paths) are introduced in Section 4.2, along with the notation $\mathrm{Alt}, \mathrm{Alt}_{\mathbf{y}}$, and $\mathrm{Alt}_{\mathbf{z}}$ (alternative arcs in a set of arcs).

6. Solution algorithm and implementation

In principle, HP could be solved by simply feeding it to an off-the-shelf MILP solver. However, we would quickly find out that even for very small instances both the number of variables and constraints would grow prohibitively large.

In order to deal with the size of the model, we apply delayed variable and constraint generation. We present the details of our custom row and column generation algorithm (Section 6.1) and and show how we separate violated constraints (Section 6.2). Finally, we show how to significantly reduce the size of the route node graph in order to speed up the constraint generation process (Section 6.3).

6.1. Delayed variable and constraint generation

Following the typical framework of a row and column generation algorithm, we build a sequence $\mathrm{HP}^0, \mathrm{HP}^1, \ldots$of subproblems of HP, where each MILP is obtained from the previous one by adding some constraints and/or some variables. The initial subProblem HP^0 is obtained from HP by removing all variables but η and all the constraints but the non-negativity constraints on η.

$$\begin{array}{lll}
\min & \sum_{u\in A}\eta_u & \\
\text{s.t.} & \eta_u \geqslant 0 & u\in A.
\end{array} \qquad \mathrm{HP}^0$$

At iteration $k = 0, 1, \ldots$ we solve Problem HP^k to optimality. Since HP^k is a relaxation of HP, if HP^k is infeasible, so is HP, and we are done.

Otherwise, let $(\eta^k, \mathbf{y}^k, \mathbf{z}^k)$ be the optimal solution to HP^k at iteration k and let $G(\mathbf{y}^k, \mathbf{z}^k)$ be the corresponding route node graph as described throughout Section 4. Then, we exploit $G(\mathbf{y}^k, \mathbf{z}^k)$ to look for violated inequalities (the details are described in Section 6.2) by performing the following steps:

1. We start by searching for strictly positive directed cycles in the graph $G(\mathbf{y}^k, \mathbf{z}^k)$, which is equivalent to searching for violated cycle inequalities. If any such cycle is found, we add the corresponding inequality to HP^k and iterate.
2. Otherwise, $G(\mathbf{y}^k, \mathbf{z}^k)$ does not contain strictly positive directed cycles and we can construct a feasible schedule $\mathbf{t}^k$ by Lemma 4, which can then be used to look for violated path inequalities. If any is found, we add it to HP^k and iterate.
3. Finally, if no cycle and path inequalities are violated, we look for capacity constraints violations. If no capacity constraints are violated, then $(\eta^k, \mathbf{y}^k, \mathbf{z}^k)$ is feasible and thus optimal for the overall Problem HP, and $\mathbf{t}^k$ is the optimal schedule by Lemma 5. Otherwise, we add the violated capacity constraint and iterate.

Note that during step 3 it might happen that HP^k does not yet contain all the variables z that are needed for separating a particular violated capacity constraint. For example, at the very first iteration, HP^0 does not contain any variable y, z, which means that no positive cycle can be generated, and the length of a path associated to a certain airplane will be simply equal to the corresponding minimum travel time. This is usually called free-running and it represents the situation in which all airplanes travel as if there were no other airplanes (see Fig. 4). The schedule $\mathbf{t}^0$ that arises from this situation might violate some of the capacity constraints, which means that some decisions must be made to prevent it. As we have seen in Section 4, these decisions are represented by the variables y, z. In general, if one of the violated capacity inequalities contains a certain variable z^s_{fg} and that variable is not present in HP^k, then we need to add to HP^k the set of variables $y^s_{fg}, y^s_{gf}, z^s_{fg}$ that is associated with z^s_{fg} together with its corresponding constraint $y^s_{fg} + y^s_{gf} + z^s_{fg} = 1$. In other words, we need to give the model the ability to prevent a certain capacity violation by adding the appropriate decision variables and constraints. However, the newly added decision variables, which are associated to newly added arcs in the route node graph (see Fig. 5), will now contribute to possibly generating more violated cycle and path inequalities.

6.2. Separating violated inequalities

The effectiveness of the solution algorithm described in the previous section is clearly conditional to the effectiveness of the procedure for separating violated inequalities. Here we describe the details of such a procedure for each class of inequality (capacity HP.*ii*, cycle HP.*iii*, and path HP.*iv*), and we show that it can be performed efficiently.

It is important to recall that the different classes of inequalities are separated sequentially. Given a vector $(\bar{\eta}, \bar{y}, \bar{z})$, with $\bar{y}, \bar{z}$ 0,1-vectors, we generate first all the violated cycle inequalities. Therefore, when we separate new classes of inequalities for the current point $(\bar{\eta}, \bar{y}, \bar{z})$, the associated graph $G(\bar{y}, \bar{z})$ contains no strictly positive directed cycles. As stated in Lemma 4, this is indeed a necessary condition for generating a feasible schedule $\bar{t}$. Also, we do not look for capacity violations until we have found a feasible schedule $\bar{t}$ that is optimal for the current subproblem, or in other words until there are no more path inequalities violated. Finding an optimal schedule is not a necessary condition for the separation of capacity inequalities, but we still separate capacity inequalities last, since this separation may require the introduction of binary variables.

- Cycle inequalities. This problem is reduced to that of finding a strictly positive directed cycle in graph $G(\bar{y}, \bar{z})$. In turn, this can be performed in $O(|V| \cdot |E|)$ time by applying a specialized label correcting algorithm (see Ahuja et al., 1993). If the algorithm returns a strictly positive directed cycle C of $G(\bar{y}, \bar{z})$, then we add the corresponding cycle inequality to the current MILP.
- Path inequalities. Let $\mathscr{P}_u(\bar{y}, \bar{z}) \subseteq \mathscr{P}_u$ be the set of *ou*-paths in $G(\bar{y}, \bar{z})$. Recall that when we separate path inequalities, $G(\bar{y}, \bar{z})$ does not contain strictly positive directed cycles. Assume that, for some $u \in A$, there exists a path inequality associated with $\bar{P} \in \mathscr{P}_u$, which is violated by $(\bar{\eta}, \bar{y}, \bar{z})$, that is

$$\bar{\eta}_u < l(\bar{P}) \left(\sum_{e \in \mathrm{Alt}_{\mathbf{y}}(\bar{P})} \bar{y}_e + \sum_{e \in \mathrm{Alt}_{\mathbf{z}}(\bar{P})} \bar{z}_e - |\mathrm{Alt}(\bar{P})| + 1 \right) \tag{17}$$

Note that $\bar{P}$ is contained in $G(\bar{y}, \bar{z})$, otherwise $\sum_{e \in \mathrm{Alt}_{\mathbf{y}}(\bar{P})} \bar{y}_e + \sum_{e \in \mathrm{Alt}_{\mathbf{z}}(\bar{P})} \bar{z}_e < |Alt(\bar{P})|$, and the r.h.s. of (17) is non-positive, a contradiction since $\bar{\eta}_u \geqslant 0$. So, $\bar{P} \in P_u(\bar{y}, \bar{z})$, and we denote by $P^*_u \in P_u(\bar{y}, \bar{z})$ the path of maximum length in $G(\bar{y}, \bar{z})$ (such path exists because the graph does not contain strictly positive directed cycles). By definition, $l(\bar{P}) \leqslant l(P^*_u)$ and thus there exists a violated path inequality if and only if

$$\bar{\eta}_u < l(\bar{P}^*_u) \tag{18}$$

So, the separation of a path inequality violated by $(\bar{\eta}, \bar{y}, \bar{z})$ amounts to computing the maximum *ou*-path in $G(\bar{y}, \bar{z})$, for $u \in A$. As for cycles, this can be performed in $O(|V| \cdot |E|)$ time. If, for some u, we have $\bar{\eta}_u < l(P^*_u)$, then we add the constraints associated with P^*_u to the current MILP. Otherwise, no violated path inequalities exist.
- Capacity constraints. When separating violated capacity constraints for point $(\bar{\eta}, \bar{y}, \bar{z})$, the associated graph $G(\bar{y}, \bar{z})$ contains no strictly positive directed cycles, and we can compute a tentative schedule $\bar{t}$ by letting $\bar{t}_u = L^*(\bar{y}, \bar{z}, u)$ for $u \in V$. Note that, since $\bar{t}$ is constructed on $G(\bar{y}, \bar{z})$, then $\bar{t}$ will respect all meetings imposed by the (arcs associated with the) meeting variables $\bar{z}$, and we can use $\bar{t}$ to check whether any of the capacity constraints are violated.
Fixed window. Checking if $\bar{t}$ violates a fixed window constraint can be done by inspection, and thus in linear time in the number of flights and the number of windows.
Sliding window. This case a bit more tricky. First, recall from (8) that if two flights f, g are in the same sliding window (with width Δ) in sector s, then we have

$$t^s_f \leqslant t^{s+1}_g + \Delta \ \wedge \ t^s_g \leqslant t^{s+1}_f + \Delta. \tag{19}$$

The above condition is satisfied if and only if the intervals $I^s_f = [t^s_f, t^{s+1}_f + \Delta]$ and $I^s_g = [t^s_g, t^{s+1}_g + \Delta]$ overlap. Consider the largest set $F^*_s \subseteq \mathscr{F}_s$ of flights mutually satisfying condition (19) in s. If $|F^*_s| > c_s$ then we have identified a violated sliding window capacity constraint (associated with F^*_s). This reduces to the problem of finding a maximum cardinality clique in an interval graph, that is the undirected graph obtained by associating a node with each interval and an edge with every pair of overlapping intervals. Finding a maximum clique in an interval graph $H = (F, W)$ can be performed in $O(|W| \log |W|)$ (see Gupta et al., 1982).

6.3. Reduction of the route node graph

The time spent separating violated path and cycle inequalities depends on the size of the route node graph. Since we use Bellman-Ford for both separations, the worst-case complexity is $O(nm)$, where n is the number of nodes and m the number of arcs.

In this section, we will show that we can look for cycles and paths in a reduced graph with only $|F| + 1$ nodes and potentially far fewer arcs than in the original route node graph. The reduced graph is quickly computed as a preprocessing step and has the potential to save significant time in each iteration of path and cycle constraint separation process. The effectiveness of the reduction has been confirmed by a set of experiments carried out during the development of our implementation. The greatest effect is expected when the number of sectors on each flight path is large.

Recall that the nodes of the route node graph correspond to the continuous schedule variables of a disjunctive formulation of the hotspot problem (see Section 2). The basic idea for our reduction is to project out all continuous-time variables except one for each flight (the arrival time $t_f^{a_f}$) and the reference time variable t_o. To this end, we use an analogue of the Fourier-Motzkin elimination scheme applied to the original disjunctive formulation. Indeed, if the formulation only contains precedence constraints and no disjunctions, then the projection operation has an immediate interpretation on the route-node graph. Namely, projecting out one variable t_v corresponds to removing the corresponding node v from the route node graph and replace each pair of arcs $(u, v), (v, w)$ (of length l_{uv}, l_{vw}, respectively), with a single arc of length $l_{uv} + l_{vw}$ (see, for instance, Mannino and Mascis, 2009). The new arc corresponds to the time precedence constraint obtained by Fourier-Motzkin combining the precedence constraints associated with (u, v) and (v, w) (see Wolsey and Nemhauser, 1999). This operation can in principle create an exponential number of constraints, and thus exponentially many arcs in the reduced graph. In case only time precedence constraints are involved in the projection, however, this does not happen. We can see this on the reduced graph, where the proliferation is prevented by the fact that when parallel arcs are created, we can remove all but the one with the largest length.

Unfortunately, when dealing with disjunctive precedence constraints, things become complicated and Fourier-Motzkin cannot be applied straightforwardly. In the following, we show one way to adapt the procedure to cope with disjunctive precedence constraints and avoid that exponentially many arcs appear. The reduced graph will however contain parallel edges. Before introducing the reduced graph $G' = (V', E')$, we need a few definitions.

Definition 6. Let f be a flight flying through sectors $s = s_1, \ldots, s_k = s'$ in sequence, where $k \geqslant 2$. Then the *flight time of f from s to s′* is

$$\delta_f(s, s') = \Lambda_f^{s_1} + \ldots + \Lambda_f^{s_{k-1}}$$

We also define $\delta_f(s', s) = -\delta_f(s, s')$.

In other words, if s precedes s' on the route of f, then $\delta_f(s, s')$ is the travel time of f from s to s', otherwise it is the negative of the travel time. In both cases, $\delta_f(s, s')$ is the length $l(P^f_{ss'})$ of the path $P^f_{ss'}$ from s to s' on the route of f in the route node graph.

Note that since $\delta_f(s, s')$ includes the flight time through sector s, but not through s', we have

$$\delta_f(s, s'') = \delta_f(s, s') + \delta_f(s', s'') \tag{20}$$

In fact, this holds regardless of the order in which f flies through s, s', and s''.

We will also use the notation $\delta(\langle f, s\rangle, \langle f, s'\rangle) = \delta_f(s, s')$ and $\delta(o, \langle f, d_f\rangle) = \Gamma_f$. We observe that $\delta(u, v) = -\delta(v, u)$ and that $\delta(u, w) = \delta(u, v) + \delta(v, w)$ if $u, v, w \in \mathscr{U}_f$ are route nodes of the same flight f, and that if G contains the edge $(\langle f, d_f\rangle, o)$, then $\delta(\langle f, d_f\rangle, o) = -\Gamma_f$.

We now go back to define the reduced graph G'. The nodes of the reduced graph are the origin o' and a node n'_f for each flight $f \in F$, corresponding to, respectively, the reference time t_o and the arrival times t_{n_f}. Note that these nodes correspond to the origin o and the arrival nodes $n_f = \langle f, a_f\rangle$ in the route node graph. The arcs of the reduced graph are of two types:

- Fixed arcs, all incident in o' (either incoming or outgoing). In particular, for every $f \in \mathscr{F}$ we have that $(o', n'_f) \in E'$. The length $\lambda(o', n'_f) = \Gamma_f + \delta_f(d_f, a_f)$ is the minimum arrival time of flight f at destination, and equals the length $l(P^f_{o,n_f})$ of the path from o to n_f in the route of f on the route node graph. When the flight f has fixed departure time Γ_f, then we also have the backward arc $(n'_f, o') \in E'$. The length $\lambda(n'_f, o') = -\Gamma_f - \delta_f(d_f, a_f) = -\lambda(o', n'_f)$ is the length of the path from n_f to o on the route of f in the route node graph.
- Alternative arcs. The alternative arcs are "copies" of the original alternative arcs, and each is associated with the same decision variable as is associated with its original version. In particular, if $e = (u, v) = (\langle f, s\rangle, \langle g, s'\rangle) \in E$ is an alternative arc of G with length l_e, then $e' = (n'_f, n'_g) \in E'$ is an alternative arc of G' and we have $\mathrm{Var}(e') = \mathrm{Var}(e)$. The length of e' is defined as

  $$\lambda_{e'} = \lambda(e') = l_e + \delta_g(s', a_g) - \delta_f(s, a_f).$$

  Note that the reduced graph may contain parallel arcs between pairs of nodes.

Definition 7 (*Reduced route node graph*). Let $G = (V, E)$ be a route node graph. The *reduced route node graph* is the graph $G' = (V', E')$, where $V' = \{o'\} \cup \{n'_f : f \in \mathscr{F}\}, E' = E'_R \cup E'_A$, and

(i) for each $f \in \mathscr{F}$, we have $e' = (o', n'_f) \in E'_R$ and $\lambda_{e'} = \Gamma_f + \delta_f(d_f, a_f)$;
(ii) for each $f \in \mathscr{F}$ with fixed departure time we have $e' = (n'_f, o') \in E'_R$, and $\lambda_{e'} = -\Gamma_f - \delta_f(d_f, a_f)$;
(iii) for each alternative arc $e = (u, v) = (\langle f, s\rangle, \langle g, s'\rangle) \in E_A$, we have $e' = (n'_f, n'_g) \in E'_A, \lambda_{e'} = l_e + \delta_g(v, a_f) - \delta_f(u, a_f)$, and $\mathrm{Var}(e') = \mathrm{Var}(e)$.

Fig. 7 shows an example of a reduced route node graph. If two flights f and g share more than one sector, then there will be multiple sets of alternative arcs between (f, a_f) and (g, a_g) in the reduced route node graph, each labelled according to the corresponding shared sector.

Lemma 8. *There is a simple path P from o to n_g in $G = (V, E)$ of length $l(P)$ if and only if there is a simple path P' from o' to n'_g in G', with $l(P) = \lambda(P')$. Moreover, the alternative arcs of P' corresponds one-to-one to the alternative arcs of P so that* $\mathrm{Var}(P') = \mathrm{Var}(P)$.

Proof. *If.* Let $e_1 = (u_1, v_1), e_2 = (u_2, v_2), \ldots, e_{q-1} = (u_{q-1}, v_{q-1})$ be the sequence of alternative arcs encountered on P, with $f_q = g$. Path P is the concatenation of $P_1 \circ e_1 \circ P_2 \circ \ldots \circ P_{q-1} \circ e_{q-1} \circ P_q$, where P_1 is the path from the origin $o = v_0$ to u_1 on the route of flight f_1, P_q is the path from v_q to $u_{q+1} = n_g$ on the route of flight $f_q = g$ and, for $1 < i < q, P_i$ is the path from route node v_{i-1} to route node u_i on the route of flight f_i (note that v_{i-1} does not necessarily precedes u_i on the route of f_i). Now, we have that $l(P) = l(P_1) + l_{e_1} + \ldots + l_{e_{q-1}} + l(P_q)$. Note that $l(P_i)$ can be negative (i.e. u_i precedes v_{i-1} on the route of f_i). By (20), for $i = 1, \ldots, q-1$ we have $l(P_i) = \delta(v_{i-1}, u_i) = \delta(v_{i-1}, n_{f_i}) - \delta(u_i, n_{f_i})$, whereas $l(P_q) = \delta(v_q, n_{f_q})$. So, we have:

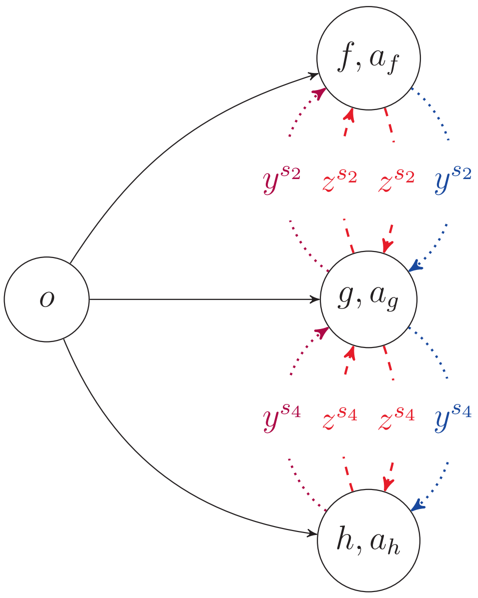


Fig. 7. Reduced route node graph with three flight f, g, and h. Flights f and g are as shown in Figure 5 and both fly through sector s_2, while flights g and h both fly through sector s_4. Flights f and h have no common sectors, so there are no alternative arcs between them. The reduced route node graph may have parallel alternative arcs, so we label the alternative arcs to distinguish them.

$$
\begin{aligned}
l(P) &= \sum_{i=1}^{q-1}\left[l_{e_i} + \delta(v_{i-1}, n_{f_i}) - \delta(u_i, n_{f_i})\right] + \delta(v_{q-1}, n_{f_q}) \\
&= \delta(v_0, n_{f_1}) + \sum_{i=1}^{q-1} l_{e_i} + \delta(v_i, n_{f_{i+1}}) - \delta(u_i, n_{f_i}) \qquad (21)
\end{aligned}
$$

where the last equality is obtained by simply rearranging the terms (i.e. taking the first term of the summation out and bringing the term outside the summation in). The above steps are illustrated in Fig. 8.

Now, the path P' on G' associated with P will be $P' = (o', n'_{f_1}) \circ (n'_{f_1}, n'_{f_2}) \circ \ldots \circ (n'_{f_{q-1}}, n'_{f_q})$, as shown in Fig. 9, and we have

$$
\begin{aligned}
\lambda(P') &= \lambda(o', n'_{f_1}) + \sum_{i=1}^{q-1} \lambda(n'_{f_i}, n'_{i+1}) \\
&= \delta(o, n_{f_1}) + \sum_{i=1}^{q-1} l_{e_i} + \delta(v_i, n_{f_{i+1}}) - \delta(u_i, n_{f_i}). \qquad (22)
\end{aligned}
$$

Only if. The proof goes like in the if case, only by reversing the construction (i.e. from P' to P).

Lemma 9. *There is a simple cycle C in $G = (V, E)$ of length $l(C)$ if and only if there is a simple cycle C' from in G', with $l(C) = \lambda(C')$. Moreover, the alternative arcs of C' corresponds one-to-one to the alternative arcs of C and $Var(C') = Var(C)$.*

The proof of the above lemma is very similar to the one of Lemma 8 and we omit it. The main idea is to prove two cases separately: one case for cycles that go through the origin, and one case for cycles that do not go through the origin.

7. Computational results

In order to test the performance of our model and our algorithm in the presence of layered capacity constraints, we run two different numerical experiments. In the first, we compare different capacity constraint combinations on a collection of small instances in order to highlight the differences in solutions and performance. In the second, we test a particular capacity constraint combination on a series of larger instances to test performance on a larger scale.

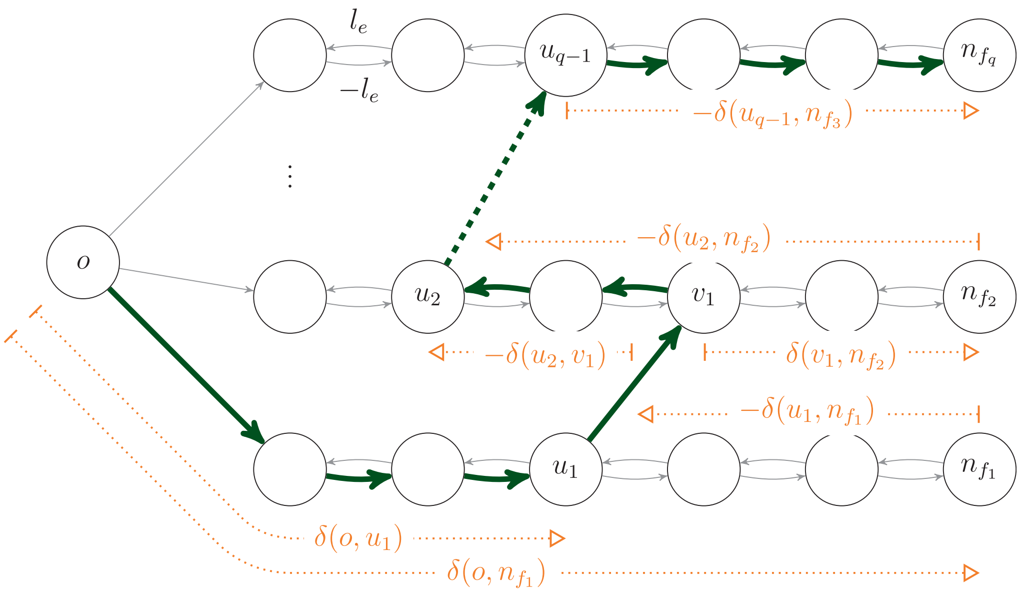


Fig. 8. A simple on_g-path P in the route node graph G. The dotted arcs represent path lengths.

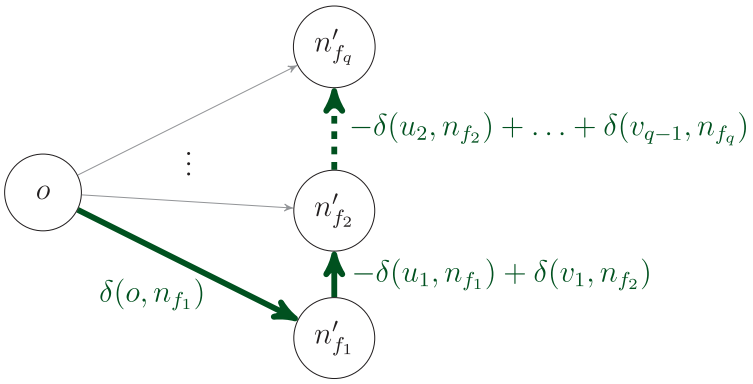

n'_{f_q}
$-\delta(u_2, n_{f_2}) + \ldots + \delta(v_{q-1}, n_{f_q})$
o
n'_{f_2}
$-\delta(u_1, n_{f_1}) + \delta(v_1, n_{f_2})$
$\delta(o, n_{f_1})$
n'_{f_1}

Fig. 9. The $on_{g'}$-path P' in the reduced route node graph G'.

Table 1
The table shows 6 different capacity constraint setups. *Baseline* represents a common existing solution, and *Goal* represents our suggested ideal solution. The alternative solutions represent various simplifications of *Goal*.

Baseline	(B)	**FW** (6, 60 min); **FW** (3, 15 min)
Alternative 1	(A1)	**FW** (6, 60 min)
Alternative 2	(A2)	**SW** (6, 60 min)
Alternative 3	(A3)	**FW** (3, 15 min)
Alternative 4	(A4)	**SW** (3, 15 min)
Alternative 5	(A5)	**FW** (6, 60 min); **SW** (3, 15 min)
Goal	(G)	**SW** (6, 60 min); **SW** (3, 15 min)

Our experiments were run on a MacBook Pro (15-inch, 2016) with a 2.9 GHz Quad-Core Intel Core i7 processor and 16 GB 2133 MHz LPDDR3 memory, running macOS Catalina version 10.15, Python 3.7.3, and CPLEX 12.9 running on a single thread with default settings.

7.1. Comparison of capacity constraint combinations

For our first experiment, we simulate traffic in a small sector. The simulated traffic follows a realistic pattern, for example for a smaller airport, and highlights the differences in performance between different capacity constraints we study. With only a few notable exceptions, most Norwegian airports have no more than 6 aircraft movements per hour on average across the year, according to 2019 data from Avinor (2020).

We simulate 7 different capacity constraint setups. The setups, summarized in Table 1, are as follows

Baseline (B). Maximum 6 flights in fixed windows of 1 h, and maximum 3 flights in fixed windows of 15 min. This solution can easily be found by conventional time-indexed formulations, so we use it as the baseline for comparison.
Alternative 1 (A1). Maximum 6 flights in fixed windows of 1 h. This is a relaxation of the baseline (B).
Alternative 2 (A2). Maximum 6 flights in sliding windows of 1 h. This is similar to (A1), but the sliding window should reduce bunching. (A2) is a strengthening of (A1), in the sense that (A1) is a relaxation of (A2).
Alternative 3 (A3). Maximum 3 flights in fixed windows of 15 min. This is a relaxation of the baseline (B).
Alternative 4 (A4). Maximum 3 flights in sliding windows of 15 min. This is similar to (A3), but the sliding window should reduce bunching. (A4) is a strengthening of (A3), in the sense that (A3) is a relaxation of (A4).
Alternative 5 (A5). Maximum 6 flights in fixed windows of 1 h, and maximum 3 flights in sliding windows of 15 min. This combination of (A1) and (A4) is a strengthening of (B). The conversion of the shorter fixed windows to sliding windows is expected to reduce any bunching introduced by the longer fixed windows.
Goal (G). Maximum 6 flights in sliding windows of 1 h, and maximum 3 flights in sliding windows of 15 min. This combination of (A2) and (A4) is a further strengthening of (A5). The use of sliding windows for both constraints is expected to further reduce bunching. However, this setup is expected to be computationally heavy.

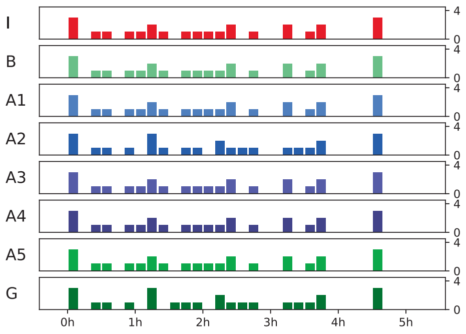
Our main interest is to compare the performances of (B), (A5), and (G). We have also included (A1), (A2), (A3), and (A4) to show the benefit of combining capacity constraints. In each of our setups using fixed windows, the fixed windows partition the timeline, with a window starting at the beginning of every hour. We use occupancy counts in each capacity constraint. The bar graphs labelled *I* in Fig. 10 show the initial traffic density in each simulated schedule.

The results of our experiments are shown in Fig. 10 and Tables 2 and 3. We have used a time cut-off of 1800 seconds[1]. The figure shows the number of flights entering the sector every 10 min. It shows that the solution for (A5) is very similar to that of (G) for the schedules where both are computed.

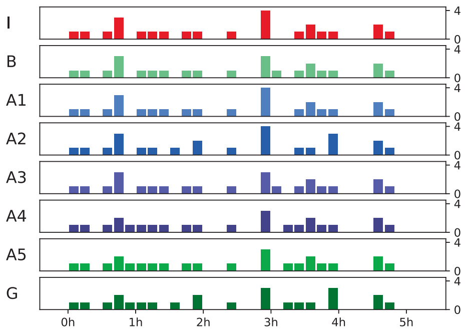
For all of the capacity constraint setups, the processing time starts growing very rapidly when the schedule becomes too crowded and too many conflicts must be resolved. Most cases that were solved before the cut-off took only a few seconds to solve, and in many cases were solved in the presolve, even after a few conflicts were added.

Looking at Fig. 10, we see that (A5) generates schedules with similar flight distributions to those generated using (G) in the cases where both schedules could be computed. That is, (A5) does not significantly increase bunching when compared to (G) in our experiments. (A4) also compares favourably to (A5) and (G) in most cases, but allows a larger sustained load since it only limits peak capacity. In our simulations, (A5) comes out as a strong com-

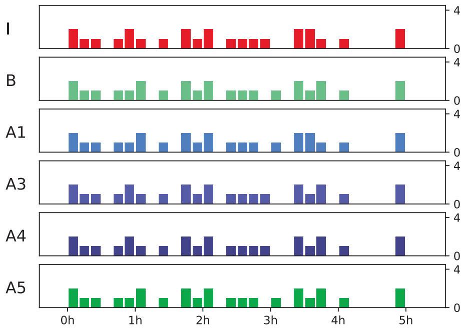
[1] Each successive MILP was solved without a time cut-off. In one case (S3, A4) an optimal solution was found after the global time cut-off. This solution has been included for comparison.



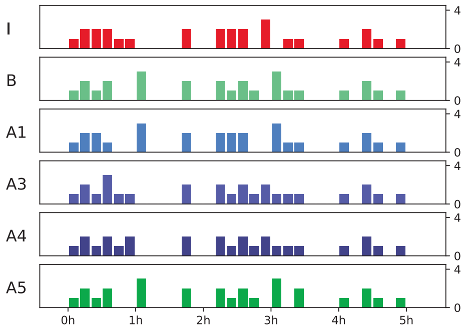
(a) S1



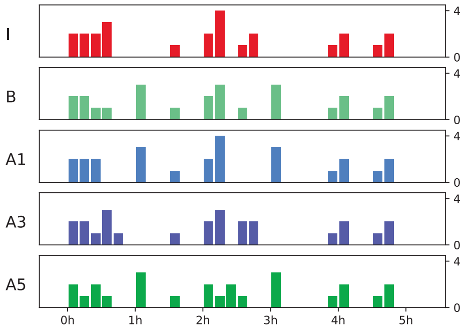
(b) S2



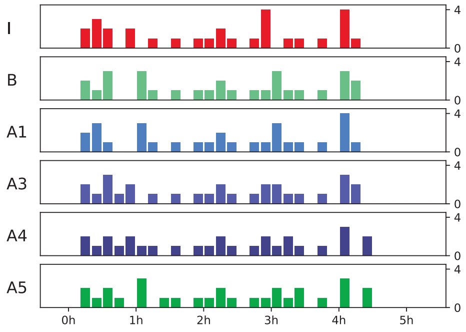
(c) S3



(d) S4



(e) S5



(f) S6

Fig. 10. An illustration of bunching. The bars show number of arrivals in 10-minute windows. The initial schedule *I* is shown in red; the baseline *B*, alternative *A5*, and the goal *G*, all combinations of two capacity constraints, are shown in shades of green; and the alternatives *A1*, *A2*, *A3*, and *A4*, all single capacity constraints, are shown in shades of blue. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

promise between lower sustained loads, less bunching, and lower processing times.

Table 3 shows the total and maximum delays in each solution. We have included maximum delays to show that our solutions do not tend to heavily delay a small number of flights, even though the maximum delay is not taken into account in our models. That is, our solutions tend to retain some measure of fairness between flights.

Tables 2 and 3 and Fig. 10 together show a fuller picture of how (A5), which is a combination of two capacity constraints, compares to a set of different single capacity constraints. The comparisons to (A1), (A2) and (G) are of particular interest. (A1) is the 1 h fixed window that is part of (A5). Since (A1) is a relaxation of (A5), it is expected to be easier to compute. However, (A5) gives significantly less bunching. (A2) is the sliding window version of (A1). (A2) is slower to compute than (A1), but does reduce bunching

Table 2
Processing time *t*, number of nodes processed *n*, and number of conflicts resolved **c**. For each schedule, we list the number of flights $|F|$. For the computations that did not complete before the processing time cut-off, we still show the number of nodes processed and conflicts resolved before the cut-off was reached. When the number of processed nodes for an instance is 0, each successive MILP for that instance was solved in the presolve. The instances are sorted according to the processing time of A5.

		B			**A1**			**A2**			**A3**			**A4**			**A5**			**G**		
	$\lvert F\rvert$	t	n	**c**	t	n	**c**	t	n	**c**	t	n	**c**	t	n	**c**	t	n	**c**	t	n	**c**
S1	25	0.0	0	**0**	0.0	0	**0**	2.1	548	**8**	0.0	0	**0**	0.0	0	**2**	0.0	0	**2**	3.4	1324	**9**
S2	24	0.0	0	**2**	0.0	0	**0**	3.1	2694	**15**	0.0	0	**2**	0.1	3	**4**	0.1	3	**4**	6.0	7083	**17**
S3	26	0.0	0	**3**	0.0	0	**2**	–	1.3e6	**63**	0.0	0	**1**	0.1	8	**6**	0.1	14	**8**	–	1.3e6	**63**
S4	27	0.1	0	**6**	0.0	0	**2**	–	1.8e6	**53**	0.0	0	**4**	21.0	4.9e4	**30**	4.0	1.4e4	**16**	–	1.8e6	**53**
S5	25	0.1	1	**6**	0.0	0	**2**	–	2.1e6	**73**	0.1	4	**6**	–	4.2e6	**59**	5.6	2.0e4	**20**	–	2.1e6	**73**
S6	29	0.1	0	**5**	0.0	0	**2**	–	1.8e6	**103**	0.0	0	**4**	2143.1	3.2e6	**54**	1159.9	2.1e6	**60**	–	1.8e6	**103**

Table 3
Total delay Σ and maximum delay *m*. The maximum delay was not taken into account by the model, but is included as a measure of how delays are distributed.

	B		**A1**		**A2**		**A3**		**A4**		**A5**		**G**	
	Σ	m	Σ	m	Σ	m	Σ	m	Σ	m	Σ	m	Σ	m
S1	**0**	0	**0**	0	**39**	13	**0**	0	**7**	5	**7**	5	**45**	13
S2	**3**	2	**0**	0	**64**	22	**3**	2	**20**	12	**20**	12	**81**	22
S3	**18**	8	**7**	6	–	–	**11**	8	**25**	12	**32**	12	–	–
S4	**75**	21	**62**	21	–	–	**15**	8	**65**	10	**107**	21	–	–
S5	**144**	25	**120**	25	–	–	**30**	11	–	–	**180**	25	–	–
S6	**59**	24	**44**	24	–	–	**29**	9	**114**	22	**133**	26	–	–

in some cases. (A5) is faster than (A2), but still appears to reduce bunching even more. In fact, (A5) seems to reduce bunching almost as much as the very much slower (G), where (G) is the standard we are aiming for.

7.2. Larger-scale performance test

In order to test the performance of our algorithm and explore its limits, we compute optimal schedules for 40 randomly generated instances with a higher traffic load. These instances all cover 6 h, with between 10 and 15 flights per hour. This traffic density range is representative for the average hourly traffic density of larger airports in Norway, like Stavanger or Bergen, according to data from Avinor (2020). The capacity constraint combination used is similar to A5, but with a fixed window capacity constraint allowing 12 entries per hour, and a sliding window constraint allowing 4 occupants in any 10-minute window. Table 4 shows the performance data.

As seen in Table 4, the number of flights in the smallest instance M1 is 63. With a sliding window capacity limit of 4, the total number of capacity constraints in the full formulation, HP, would be $\binom{63}{5}$, which is just over 7 million constraints in almost 2 thousand binary variables. Had the hourly limit of 12 flight also been a sliding window, the total number of capacity constraints in the full HP would be over 10^{13}. As shown in the table, the number of capacity constraints generated is only a tiny fraction of the total number of such constraints. Furthermore, since every permutation (down to rotational symmetry) of every subset of the flights corresponds to a cycle, the number of cycle constraints in the full HP would be $62! + 61! + \ldots$, which is about 10^{85}. Computing the number of path inequalities would require an in-depth analysis of the individual instance. A comparison of t_{total} and t_{last} shows that a significant amount of the total processing time is spent solving the final formulation $\text{HP}^{k_{\text{last}}}$. Since the final formulation can be very small compared with the total formulation, the overhead in processing time from delayed variable and constraint generation is comparatively small.

Figs. 11 and 12 show how the total processing time varies with other solution statistics. The size of each dot represents the number of flights in the related instance, but not in direct proportion. Differing sizes are used to show that total processing time is affected by more than just the number of flights in the instance.

Fig. 11 shows how the total processing time varies with the number of time the MIP solver is invoked, and the total number of MIP nodes processed. We use a logarithmic scale for total processing time since the time tends to increase exponentially with some measure of instance size. In Fig. 11 we use log-log axes since the quantities being compared appear to have a near-linear relationship.

Fig. 12 shows how the total processing time varies with the number of binary variables and the number of generated constraints. In this case, we use log-linear axes, since the total processing time appears to increase exponentially with the number of variables, and near exponentially in the number of constraints.

As we can see in Table 4, our model solves most instances very quickly and only breaks down when the traffic becomes very dense and the number of separated capacity conflicts exceeds a few dozen. Since the schedules we are targeting are ones initially created to be handled by controllers, we can reasonably expect that real-world instances will have a relatively small number of capacity conflicts to be resolved.

8. Conclusions and future work

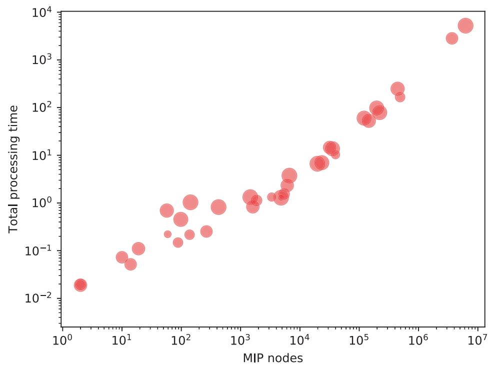
We have shown that the Path&Cycle approach can be used to model the Hotspot Problem with occupancy counts and layered capacity constraints. We have also discussed how we can speed up the solution process so that we can use our model to solve instances with novel capacity constraint setups that are of interest to air-traffic control authorities.

It is clear from the model HP that sliding window capacity constraints can generate a very large number of variables, should the number of conflicts grows too large. This has led us to propose (A5), where fixed windows are used to spread the flights out, and a sliding window is used to smooth away bunching. With access to real-world instances, we could further validate our approach and test its scalability under a variety of conditions.

Table 4 shows how our model performs on a collection of randomly generated instances with a larger number of flights. We

Table 4
Computational performance on 40 randomly generates instances. In each case, the traffic is divided across 6 h, with between 10 and 15 flight per hour. The capacity constraint is similar to A5, with at most 12 entries per hour, and at most 4 occupants in any 10-minute window. The instances are sorted according to the total number of flights and total processing time. Each instance was run with a cut-off where no new calls were made to the MIP solver after 1 h. Instance M22 has a node count of 0 because it was solved in the initial presolve.

	$\|F\|$	MIP nodes	MIP solves	t_{total}	t_{last}	t_{callback}	n_{vars}	n_{capacity}	n_{path}	n_{cycle}
M1	63	4805	12	1.29	0.30	0.05	90	11	190	6
M2	64	59	6	0.22	0.05	0.03	57	5	129	5
M3	65	3340	7	1.33	0.43	0.05	117	6	252	0
M4	65	39849	23	10.30	3.98	0.24	153	22	432	55
M5	66	88	4	0.15	0.08	0.03	72	3	142	1
M6	66	138	4	0.22	0.13	0.03	90	3	159	0
M7	66	488582	139	165.82	4.05	1.18	240	138	576	147
M8	67	2	2	0.02	0.02	0.01	30	1	54	0
M9	67	1858	9	1.13	0.57	0.09	111	8	324	20
M10	67	5429	12	1.55	0.76	0.09	120	11	257	12
M11	68	14	4	0.05	0.02	0.02	45	3	100	0
M12	68	10	3	0.07	0.05	0.02	60	2	109	0
M13	68	266	4	0.25	0.16	0.03	90	3	163	0
M14	68	3663575	163	2829.23	329.72	2.71	285	162	1312	427
M15	68	6178644	45	–	–	–	210	45	1149	410
M16	68	8196053	38	–	–	–	219	38	815	82
M17	69	2	2	0.02	0.02	0.01	30	1	54	0
M18	69	19	3	0.11	0.08	0.03	60	2	108	0
M19	69	1606	9	0.83	0.51	0.08	132	8	353	4
M20	69	6118	8	2.33	1.45	0.15	219	7	401	13
M21	69	31692	29	14.57	1.12	0.89	162	28	373	222
M22	70	0	1	0.00	0.00	0.00	0	0	0	0
M23	70	145229	20	52.90	32.43	0.46	303	19	735	74
M24	70	443948	20	247.86	38.50	1.62	213	19	1132	343
M25	71	57	20	0.69	0.10	0.40	240	19	279	80
M26	71	23364	11	7.00	5.10	0.19	204	10	415	23
M27	71	35420	71	13.72	3.43	0.71	309	70	419	101
M28	71	120844	27	60.08	20.77	2.44	321	26	603	450
M29	71	221104	27	78.66	41.22	0.52	177	26	695	125
M30	71	196980	24	97.81	62.80	0.79	348	23	807	135
M31	71	4296321	125	–	–	–	468	125	1251	526
M32	72	98	9	0.45	0.08	0.19	153	8	171	48
M33	72	425	15	0.82	0.32	0.54	243	14	242	108
M34	72	143	16	1.03	0.21	0.52	252	15	297	100
M35	72	1457	17	1.33	0.26	0.29	219	16	309	50
M36	72	6649	19	3.75	1.22	0.57	300	18	422	95
M37	72	19707	13	6.64	1.96	0.26	216	12	657	41
M38	72	6212160	55	5222.85	1955.66	1.26	282	54	1158	229
M39	73	9541090	295	–	–	–	339	295	873	186
M40	73	7633255	420	–	–	–	423	420	665	272



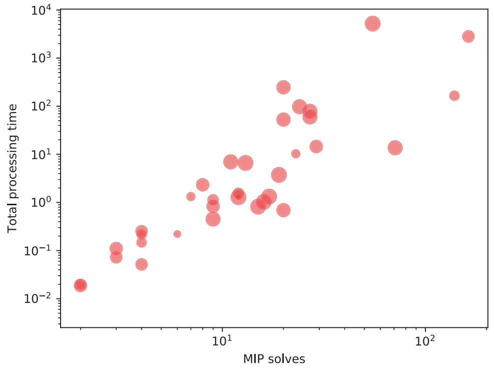


Fig. 11. Total processing time against, respectively, the total number of MIP nodes and the number of calls to the MIP solver. Since the running time appears to increase exponentially with some measure of the difficulty of the instance, the processing time is shown on a logarithmic scale. Since the processing time is approximately linear in the number of MIP nodes and solves respectively, these are also shown on a logarithmic scale. The size of each dot represents the number of flights in the instance, but not in direct proportion.

see that even for quite dense traffic patterns, a relatively small number of variables and constraints are generated before an optimal solution is found. Figs. 11 and 12 show that the total processing time of an instance depends more on the number of conflicts (variables and constraints) detected than on the number of flights in the instance. This is in line with the findings in

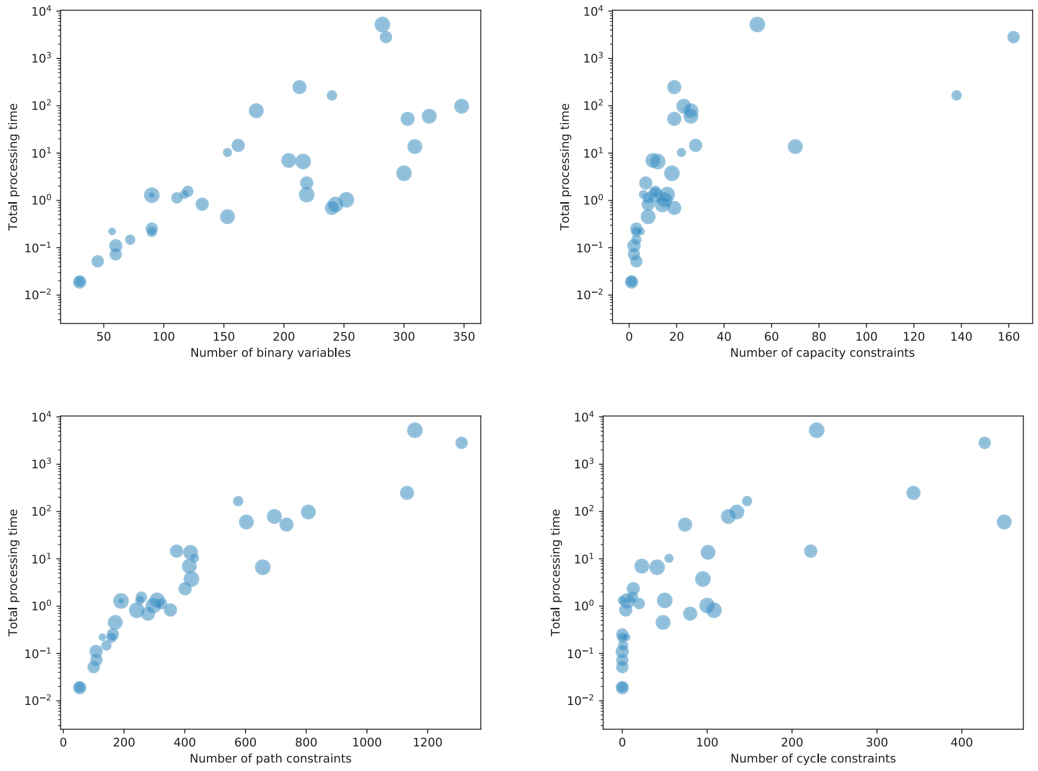


Fig. 12. Total processing time against respectively, the number of variables, capacity constraints, path constraints, and cycle constraints. Since the running time appears to increase exponentially with some measure of the difficulty of the instance, the processing time is shown on a logarithmic scale, while the number of variables and constraints are shown on a linear scale. The size of each dot represents the number of flights in the instance, but not in direct proportion.

Mannino and Sartor (2018), where the Path&Cycle approach is used to solve a version of the Hotspot problem for multiple, low-capacity sectors. This scaling property makes the Path&Cycle approach well suited to last-minute Air Traffic Flow Management, where the current schedule in nearly feasible.

In the future, we would like to build a more precise understanding of the problems surrounding bunching. In order to show bunching effects, we use Fig. 10 to illustrate the density of flights in 10-minute intervals. This gives a good visual indication of the bunching effect, and studying the figure can give us an indication about how to tweak our capacity constraints. Working with controllers and control authorities we could further our understanding of how bunching affects the workload on controllers, which in turn would allow us to further improve our analysis of different capacity constraint setups.

The Hotspot Problem as we define it falls under last-minute Air Traffic Flow Management (ATFM), which does not take into account aircraft separation and other local feasibility constraints. Instead, capacities are set low enough that local Air Traffic Control (ATC) can handle feasibility in each sector of the airspace. The separation of ATFM and ATC requires ATFM to be very conservative. By expanding our model into the domain of ATC, we could potentially allow higher capacities in the ATFM problem by enforcing local feasibility already at the ATFM level. Our model can already handle temporal aircraft separation constraints, so the major challenge would be to expand the model to route and flight level selection. In this case, we may have to resort to heuristic approaches, like the ones presented in Kim et al. (2009), Samà et al. (2017), or other tried approaches for job-shop scheduling (Allahverdi, 2016).

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Carlo Mannino: Conceptualization, Methodolody, Writing - review & editing, Supervision, Funding acquisition. **Andreas Nakkerud:** Methodology, Software, Writing - original draft, Writing - review & editing, Visualization. **Giorgio Sartor:** Conceptualization, Methodolody, Writing - review & editing.

References

Ahuja, Ravindra K., Magnanti, Thomas L., Orlin, James B., 1993. Network Flows: Theory, Algorithms, and Applications. Prentice-Hall Inc., USA.
Allahverdi, Ali, 2016. A survey of scheduling problems with no-wait in process. Eur. J. Oper. Res. 255 (3), 665–686.
Allignol, Cyril, Barnier, Nicolas, Flener, Pierre, Pearson, Justin, 2012. Constraint programming for air traffic management: a survey: in memory of pascal brisset. Knowl. Eng. Rev. 27 (3), 361–392.

Avella, Pasquale, Boccia, Maurizio, Mannino, Carlo, Vasilyev, Igor, 2017. Time-indexed formulations for the runway scheduling problem. Transp. Sci. 51 (4), 1196–1209.

Avinor, 2020. Avinor Statistics Archive. https://avinor.no/en/corporate/about-us/statistics/archive, (visited September 2020)..

Bertsimas, Dimitris, Tsitsiklis, John N., 1997. Introduction to Linear Optimization. Athena Scientific Belmont, MA.

Bianco, Lucio, Dell'Olmo, Paolo, Giordani, Stefano, 2006. Scheduling models for air traffic control in terminal areas. J. Schedul. 9, 223–253.

Damhuis, E.J.H., Visser, H.G., de Jonge, Hugo.W.G., Seljée, Ron.R., 2015. Optimising air traffic flow management. Technical Report NLR-TP-2015-180, National Aerospace Laboratory NLR, 2015..

D'Ariano, Andrea, Pacciarelli, Dario, Pistelli, Marco, Pranzo, Marco, 2015. Real-time scheduling of aircraft arrivals and departures in a terminal maneuvering area. Networks 65 (3), 212–227.

de Jonge, Hugo.W.G., Seljée, Ron.R., 2011. Optimisation and Prioritisation of Flows of Air Traffic through an ATM Network. Technical Report NLR-TP-2011-567, National Aerospace Laboratory NLR..

Dubot, Thomas, Bedouet, Judicaël, Degrémont, Stéphane, 2016. Modelling, generating and evaluating sector configuration plans. In: 30th Congress of the International Council of the Aeronautical Sciences (ICAS 2016).

Geraldine Flynn, Benkouar, A., Christien, R., 2003. Pessimistic sector capacity estimation. Technical Report EEC Note No. 21/03, Eurocontrol Experimental Centre..

Guibert, S., Fitzpatrick, M., Criscuolo, P., Dohy, D., Iliev, B., Allard, E., Fabio, A., Puntero, E., Iglesias, E., Carrera, T., Valle, N., Neyns, V., Karahasanovic, A., 2019. SESAR Solution 08.01 Validation Report (VALR) for V2. Technical Report D2.1.050, SESAR Joint Undertaking..

Gupta, Udaiprakash I., Lee, Der-Tsai, Leung, Joseph Y.-T., 1982. Efficient algorithms for interval graphs and circular-arc graphs. Networks 12 (4), 459–467.

Joondong Kim, Alexander Kroeller, Mitchell, Joseph S.B., 2009. Scheduling aircraft to reduce controller workload. In: Jens Clausen, Gabriele Di Stefano (Eds.), ATMOS 2009 – 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, IT University of Copenhagen, Denmark, September 10, 2009, Volume 12 of OASICS. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

Lamorgese, Leonardo, Mannino, Carlo, 2019. A noncompact formulation for job-shop scheduling problems in traffic management. Oper. Res. 67 (6), 1586–1609.

Mannino, Carlo, Mascis, Alessandro, 2009. Optimal real-time traffic control in metro stations. Oper. Res. 57 (4), 1026–1039.

Mannino, Carlo, Nakkerud, Andreas, Sartor, Giorgio, Schittekat, Patrick, 2018. Hotspot resolution with sliding window capacity constraints using the Path&Cycle algorithm. SESAR Innov. Days.

Carlo Mannino, Giorgio Sartor, 2018. The Path&Cycle formulation for the hotspot problem in air traffic management. In: Ralf Borndörfer and Sabine Storandt, editors, 18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018), volume 65 of OpenAccess Series in Informatics (OASIcs), pp. 14:1–14:11, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik..

Mascis, Alessandro, Pacciarelli, Dario, 2002. Job-shop scheduling with blocking and no-wait constraints. Eur. J. Oper. Res. 143 (3), 498–517.

Maurice Queyranne, Schulz, Andreas S., 1994. Polyhedral approaches to machine scheduling. Technical Report 408/1994, Technische Universitat Berlin..

Tolebi Sailauov, Zhao Wei Zhong, 2016. An optimization model for large scale airspace. Int. J. Model. Optim. 6 (2), 86..

Samà, Marcella, D'Ariano, Andrea, Corman, Francesco, Pacciarelli, Dario, 2017. Metaheuristics for efficient aircraft scheduling and re-routing at busy terminal control areas. Transp. Res. C Emerg. Technol. 80, 485–511.

Nina Schefers, Miquel Angel Piera, Juan José Ramos, Jenaro Nosedal, 2017. Causal analysis of airline trajectory preferences to improve airspace capacity. Procedia Comput. Sci. 104, 321–328..

SESAR Joint Undertaking, 2020. Single European Sky ATM Research (SESAR) Joint Undertaking. https://www.sesarju.eu, (visited April 2020)..

Vaaben, Bo, Larsen, Jesper, 2015. Mitigation of airspace congestion impact on airline networks. J. Air Transp. Manage. 47, 54–65.

Wolsey, Laurence A., Nemhauser, George L., 1999. Integer and Combinatorial Optimization, vol. 55. John Wiley & Sons.

Zhao Wei Zhong, 2018. Overview of recent developments in modelling and simulations for analyses of airspace structures and traffic flows. Adv. Mech. Eng. 10 (2)..

Paper III

Optimal Train Rescheduling in Oslo Central Station

Carlo Mannino, Andreas Nakkerud

Submitted to *Omega,* revision requested by editor.

Optimal Train Rescheduling in Oslo Central Station[⋆]

Carlo Mannino[b,a], Andreas Nakkerud[a,b,*]

[a]*Department of Mathematics, P.O box 1053 Blindern, 0316 OSLO, Norway*
[b]*SINTEF, P.O box 124 Blindern, 0314 OSLO, Norway*

Abstract

Real-time train dispatching (i.e., rescheduling and replatforming) in passenger railway stations is a very important and very challenging task. In most major stations, this task is carried out by hand by highly trained dispatchers who use their extensive experience to find near-optimal solutions under most conditions. Under major disruptions, however, the traffic situation may become too complex for any human to handle it far beyond finding feasible solutions. As part of a prototype for a dispatching support tool developed in collaboration with Bane NOR (Norwegian rail manager), we develop an approach for Optimal Train Rescheduling in large passenger stations. To allow for replatforming, we extend the standard job-shop scheduling approach to train-scheduling, and we develop and compare different MILP formulations for this extended approach. With this approach, we can find, in just a few seconds, optimal plans for our realistic instances from Oslo Central Station, the largest passenger train hub in Norway. The prototype will be tested by dispatchers in the greater Oslo area, starting from the fall of 2021.

Keywords: Integer programming, Optimization, Rail transport, Dispatching, Scheduling, Routing
2010 MSC: 90B06, 90B20, 90C06, 90C08, 90C11, 90C90

1. Introduction

Like all management of critical infrastructure, train dispatching is heavily regulated. Under the current system, all dispatching decisions must be made by highly trained human dispatchers. Therefore, the only practical way to introduce optimization into the process is through decision support tools. This work is part of the GOTO project [18] with Norwegian rail manager Bane NOR. The GOTO project aims to deliver an optimization-based decision support tool for dispatching trains in Oslo Central Station and other large passenger train stations. While the tool we develop is aimed at Oslo Central Station, the algorithms we present are general and not tailored to this station. The layout of Oslo Central Station (Figure 10) is typical of large passenger train stations.

[⋆]This work was funded by The Research Council of Norway [grant numbers 237718, 267554, 300509].
[*]Corresponding author
Email address: `andreana@math.uio.no` (Andreas Nakkerud)

In order to have a decision support tool accepted, we must make sure that our approach can at least match or, better, outperform the human dispatchers under normal traffic conditions. Under these conditions, human dispatchers can use their expert intuition to evaluate suggested solutions and compare them to the near-perfect solutions they produce. Based on evaluations under normal traffic conditions, dispatchers may come to trust the suggested solutions in heavily congested traffic situations where no human can expect to capture the complete picture. It follows that we must model the infrastructure and business rules with a very high level of detail to produce high-quality solutions under any conditions.

The need for automated decision support can only be expected to grow with the introduction of new technology in railway signaling. Currently, almost all dispatching is based on a fixed division of the track infrastructure by signals. Level 3 of the European Train Control System (ETCS) introduces *moving blocks* (see [19]), where trains are protected by safe zones determined by breaking distances rather than by signals at fixed locations. As these new control systems are introduced, they will increase the flexibility of train rescheduling and make it even harder to solve the train dispatching problem optimally by hand. In order to make full use of the flexibility introduced by moving blocks, we will require fine-grained scheduling approaches.

An extensive research effort has gone into real-time train rescheduling problems (see, for example, survey papers [4, 6, 8]), but the research primarily covers simple railway network designs. Only a few works are devoted explicitly to dispatching trains in (large) railway stations. The works typically make use of Mixed Integer Linear Programming (MILP) formulations [20]. We can identify two major classifications according to the way scheduling and platforming (routing within the station area) are represented in the models.

- For the scheduling part, two main streams of MILP models are applied in the literature: *big-M* formulations and *time-indexed* formulations [16]. In both models, the path of a train through the stations or lines is subdivided into smaller segments (sometimes down to the physical *track circuits*, which are the smallest regions in the train detection systems). In big-M formulations, for each train and each segment in its path, we have a continuous variable representing the time in which the train enters the segment. The drawback of this approach is that we need to introduce a disjunctive constraint to represent the order in which two trains travel through a contended track. These, in turn, are translated into linear constraints by introducing binary variables and the so-called big-M constraints, i.e., constraints containing some very large coefficients–notoriously weakening the formulation [16]. In time-indexed models, the planning horizon is discretized into small time periods. A binary variable is associated with each train, each segment in its path, and time period. The resulting formulations are typically stronger than their big-M counterparts, but they have a much larger number of variables and constraints, slowing down the solution process. The smaller the time period, the larger the number of variables: on the other hand, large time periods lead to a poor approximation of the train movement through the station, which may end up generating suboptimal solutions, or even in solutions that cannot be implemented in practice [10].

- For the platforming part, we can identify two major categories according to how paths through the station are represented. In *multicommodity flow* approaches [1], a binary variable x is associated with each train and each segment of its path, and $x = 1$ denotes that the train will run through the segment. In this class of approaches, the train path is constructed directly by the model. The drawback of this approach is that the model must incorporate flow constraints to represent paths through the station. In *path-based* approaches, we have a binary variable associated with each train and each potential path of the train through the station. The drawback of this approach is that the number of paths may grow exponentially with the station's size.
  Finally, the two approaches may be combined using Dantzig-Wolfe decomposition and column generation (see [7]). With this technique, a path-based MILP is constructed iteratively by solving a sequence of single-commodity flow subproblems.

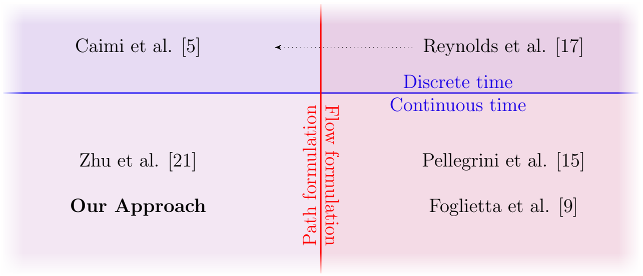


Figure 1: Categorization of some recent papers on replatforming and rescheduling in railway stations.

Figure 1 shows how some recent papers on the topic are divided according to the classifications described above.

Outside these major classes of approaches, there are some simulation-based and heuristic approaches.

Reynolds et al. [17] present a time-indexed multicommodity flow model for rescheduling and replatforming. In their approach, they then transform their formulation into a path-based one by Dantzig-Wolfe decomposition. They then solve this formulation by branch-and-price and column generation [2]. Reynolds et al. apply their approach to solve instances with up to 32 trains (1 hour of traffic) in a large station area. They use a 15-second time step for their time-indexed formulation and 30-second margins to account for business rules that their model does not take into account.

Caimi et al. [5] present a discrete-time path-based formulation for rescheduling and replatforming. They introduce blocking stairways, which detail the speed profile of a train and when the train blocks different segments. The trains are then assigned to available blocking stairways for entry to and exit from the station. If no feasible solution can be found, additional blocking stairways are generated in a column generation fashion. The blocking stairways allow the track infrastructure to be modeled to the level of track circuits, but many blocking stairways may need to be generated. Caimi et al. apply their approach to instances of the central station of Berne, Switzerland, where they solve a whole operational day (roughly 1500 trains) in about 1$^1/_2$ hours.

Pellegrini et al. [15] present a continuous-time multicommodity-flow-like approach. In their approach, like in that of Raynolds et al. [17], the routing is left to the solver. They also model the track infrastructure at the level of track circuits, which is the highest available resolution for train position detection in most current signaling systems. Pellegrini et al. apply their approach to Lille-Flandres station, where they solve instances with up to 47 trains, but only for up to 450-second (7$^1/_2$-minute or $^1/_8$-hour) periods.

Zhu et al. [21] present a continuous-time path-based formulation similar to the one we present here. However, their formulation is slightly simplified and is used to solve smaller instances in order to support an overarching agent-based approach. Zhu et al. show detailed analyses of computational results for MILP instances with four trains. He et al. [11] also present a similar path-based formulation, but they use it as part of a simulation-based approach rather than as a MILP formulation.

To our knowledge, although tested on real-life or realistic instances, none of the above approaches have been implemented in control centers and tested or adopted by operative dispatchers. Foglietta et al. [9] present a *heuristic* approach that was in operation to support dispatchers in Roma Tiburtina. While their paper also describes an exact, flow-based IP model, this model required a commercial solver and was not applied in the station.

As this paper is part of the research project GOTO in collaboration with Bane NOR, our modeling requirements have been guided by the use-cases at Bane NOR. In particular, we want to focus on real-time dispatching support for Oslo Central Station. This station acts as a hub, connecting traffic bound for the south-east and the south of Sweden; for the east and central Sweden; for the north; for the south-west; and for the west. Delays in Oslo Central Station can have knock-on effects on the entire Norwegian rail network, both for passengers and cargo. In order to control these effects, we require a very long planning horizon.

Trains entering and leaving Oslo Central Station must be highly coordinated since they often share track resources. To avoid causing unnecessary delay, we must model the infrastructure and business rules of the station very accurately, and we must avoid adding catch-all buffer times. In late 2019, the Director for Customers and Traffic of Bane NOR, the Norwegian rail manager, told Norwegian newspaper Aftenposten [3] that to increase punctuality for 2020, trains will close their doors 20 seconds before their scheduled departure. This statement indicates that the flow of passenger-train traffic is sensitive to very small delays and that we must aim for a very fine time resolution.

In order to achieve sufficient accuracy, we model the track infrastructure on the level of track circuits [19], which offer the finest resolution of train location in fixed-signal based train control. Then, we extend the alternative-graph big-M formulations of Mannino et al. [12, 13] to include path selection, and we develop continuous-time formulations for the train dispatching problem in the large, hub-like passenger station Oslo Central Station in Norway.

We adopt the path-based approach for platforming and assign collections of possible paths to each train. Candidate paths are pre-selected based on observations of traffic and

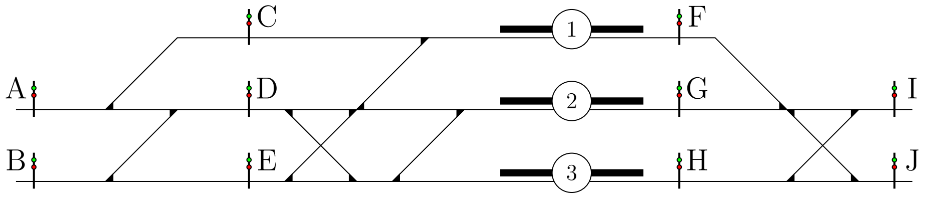


Figure 2: We represent a station by a set of signals and a set of interlocking routes connecting pairs of signals. All train movements in the depicted area is from left to right.

discussions with dispatchers in Oslo Central Station. As a result, we can offer the flexibility in routing expected by dispatchers while also finding optimal dispatching solutions in a reasonable time for real-time applications. Using 1 hour of observed rush-hour traffic, we craft instances of 6 hours of rush-hour traffic (330 trains) and instances with 24 hours of traffic with morning and afternoon rush-hours (996 trains). In both cases, we solve all instances to optimality in a reasonable amount of time for real-time applications in dispatching.

A support tool built on our algorithm will be field-tested by dispatchers at Oslo Central Station, starting in the fall of 2021. This field test is part of the ongoing GOTO project, which has delivered a line dispatching prototype already in active testing on the lines incident to the station.

2. The Optimal Dispatching Problem

In this section, we give a formal description of the optimization problem tackled in this paper. We consider the simultaneous rescheduling and replatforming of passenger-train traffic through large passenger stations. The combination of rescheduling and replatforming is the typical task of dispatchers. The Optimal Dispatching Problem (ODP) is the task of assigning tracks and schedules to trains in a way that minimizes delays or maximizes passenger utility. In this paper, we aim to minimize the (weighted) sum of delays for all trains. For our computational experiments, we solve ODP for Oslo Central Station, the largest hub for passenger-train traffic in Norway. We consider a scheduling horizon of up to 24 hours.

2.1. Track Infrastructure: Signals and Interlocking Routes

On the most basic level of scheduling, we represent the track infrastructure of a station as a set of signals and a set of *interlocking routes*, which are the track sections connecting two successive signals. The movement of a train can be decomposed into a sequence of elementary movements, one for each interlocking route of its path. This decomposition is of particular practical interest since, under normal operations, the interlocking routes are at the highest level of precision in scheduling train movements in signal-based train control systems [19]; dispatchers control trains on the level of signals.

Figure 2 shows an example of a station with three platforms. Often, an interlocking route is uniquely determined by the signals it connects, but not always. In Figure 2, there are

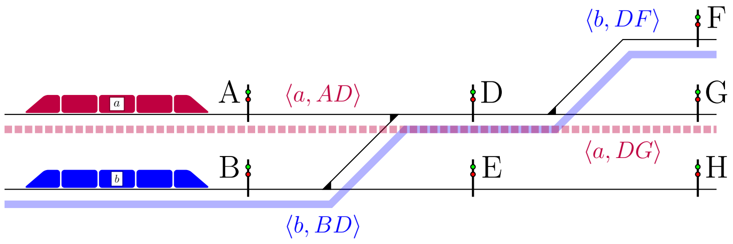


Figure 3: Train a (purple) is about to enter interlocking route AD, and train b (blue) is about to enter interlocking route BD. Since the two interlocking routes physically overlap, one of the trains must wait.

two possible interlocking routes connecting signals D and G. An interlocking route cannot pass a signal, so the remaining interlocking routes are uniquely defined by the signals they connect. Signals are directional, and signals for opposite directions need not be placed at the same point along the tracks. Therefore, there need not be any correspondence between interlocking routes in opposite directions.

2.2. Paths and Station Platform Tracks

The station in Figure 2 has three platforms, drawn as solid, black rectangles with a circle containing the track number. Each train passing the station will have a set of permitted paths through the station, where a path is a sequence of interlocking routes. One or more paths may be preferred for a specific train, for example, paths using one of the tracks adjacent to a given platform. We may associate a cost with the choice of path.

In principle, any physically connected sequence of interlocking routes can be a feasible path. Usually, however, only a few paths are actually available to a given train because of business rules and other operational considerations. In a large station like Oslo Central Station, it is typically required that a train stops at its designated track or, possibly, the track opposite on the same physical platform.

2.3. Timetable and Delays

For each train, we are given a set of stations where the train is supposed to stop and the scheduled arrival and departure times at these stations. Together, these are referred to as the timetable. Given a station, our task is to decide, for each train, which path it will take and when it will pass each signal on its path. In general, we may be given a scheduled arrival time and an earliest departure time for any signal.

When we reschedule, the *delay* of a train in a station is the difference between the (re-)scheduled arrival time and the arrival time in the timetable, or 0 if the difference is negative. We assume that the timetable is independent of the choice of path through the station. That is, scheduled arrival and departure times do not depend on the choice of platform track.

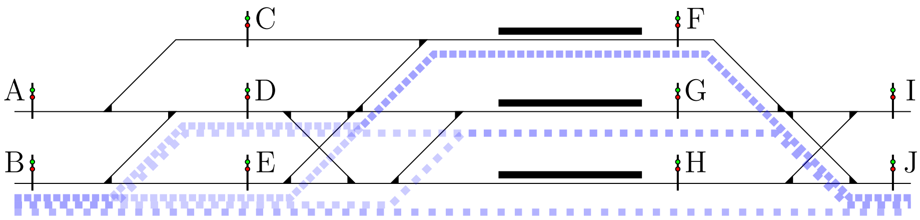


Figure 4: An example collection of paths for a train going through the station from signal A to signal J. Not all possible paths are included in the collection.

2.4. Scheduling Conflicts

When two trains are set to use the same interlocking route (or a pair of physically overlapping interlocking routes), we have a *potential scheduling conflict* as shown in Figure 3. In Figure 3 we identify two potential scheduling conflicts. The first is between train a entering interlocking route AD and train b entering interlocking route BD, and the other is between a entering DG and b entering DF. Generally, a pair of interlocking routes is *incompatible* if some physical restriction or business rule limits their simultaneous use. An interlocking route will always be incompatible with itself.

A potential conflict is *realized* by a given schedule only if, according to the schedule, the two trains *simultaneously* occupy the interlocking route(s) generating the conflict. It is apparent that if a schedule realizes a conflict, then the schedule is not feasible. We will use the term *candidate schedule* when we want to emphasize that the schedule may be infeasible.

3. The Model

3.1. Route Nodes and Schedules

As discussed in the previous section, and illustrated in Figures 2 and 3, the movement of a train can be decomposed into a sequence of elementary movements, each through an interlocking route of the train's path. We assume the speed of a train to be constant through an interlocking route so that movement can be described by the entry time of the train in each interlocking route of its path. We assume unique entry signals to and exit signals from the modeled area for each train.

Figure 4 shows a collection of five paths from signal B to signal J through the station in Figure 2. Each path consists of three interlocking routes. Figure 5 is a graph representation of how the interlocking routes in Figure 4 are connected into paths through the station. Each directed edge in Figure 5 represents a permitted transition from one interlocking route to another, and each directed path in the figure represents an available path through the station. The nodes labeled XB and JY are the entry and exit interlocking routes, respectively. Note that Figures 4 and 5 represent the same possible collection of path options available to a train passing the station. Different trains may have different path options, and the picture in the figures is not a complete representation of all the path options in the station.

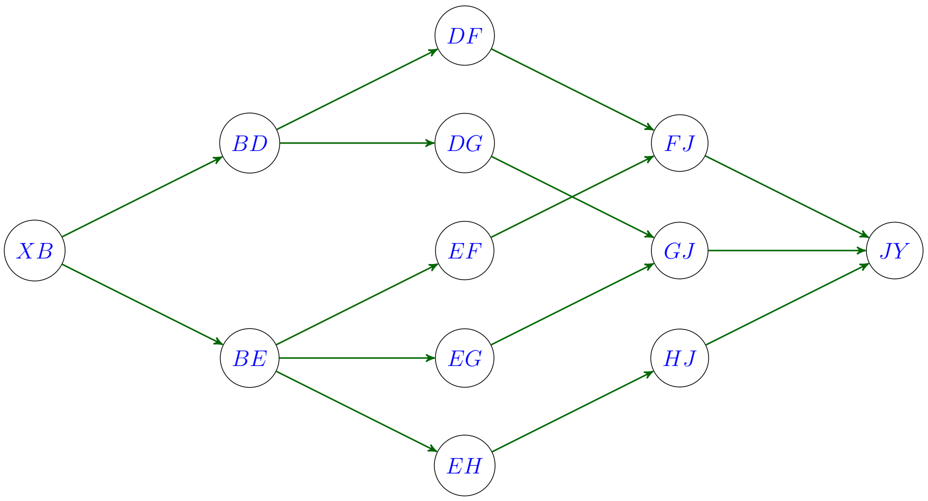


Figure 5: The figure shows the directed graph associated with the interlocking routes of the paths highlighted in Figure 4. XB is the entry interlocking route leading to signal B, and JY is the exit interlocking route leading away from signal J.

For each train a, and each interlocking route r that may be used by a, we define the *route node* $\langle a, r\rangle$. We let $\mathcal{N}(a)$ be the set of all route nodes of train a. Informally, a route node $\langle a, r\rangle \in \mathcal{N}(a)$ represents the occupation of the interlocking route r by train a. A path for a through the station corresponds then to a subset of route nodes in $\mathcal{N}(a)$.

We let $\mathcal{A}$ be the set of all trains, and define $\mathcal{N} = \{o\} \cup \bigcup_{a\in\mathcal{A}} \mathcal{N}(a)$. That is, $\mathcal{N}$ is the set of all route nodes associated with the trains, plus a special node o which represents the origin of the planning horizon. We let $\mathcal{N}_S \subset \mathcal{N}$ be the set of sink (or terminal) route nodes, i.e., the route nodes representing the end of the journey of a train (in the modeled area).

A *schedule* is a function $\mathbf{t} : \mathcal{N} \to \mathbb{R}$. We let $t_u = \mathbf{t}(u)$. A schedule associates a time to each route node, and $t_{\langle a,r\rangle}$ is the time train a enters interlocking route r, if r belongs to the path chosen for a. Note that, since there are alternative paths available, an interlocking route r available for a may not be chosen. In this case, $t_{\langle a,r\rangle}$ may assume any value.

The time t_o associated with the origin is the start time of our planning horizon, and we have

$$t_u \geq t_o \qquad\qquad u \in \mathcal{N} \tag{1}$$

For ease of explanation, through this section, we assume that the path through the station, i.e., the sequence of interlocking routes, is fixed in advance for any train a. In this case, the graph of available interlocking routes (Figure 5) reduces to an oriented, simple path and the set $\mathcal{N}(a)$ to the nodes in this path.

3.2. Release Times and Free Running

When we consider the schedule of an individual train in isolation, without any interaction with other trains, we say we are considering the *free running* of the train. In free running, a train's schedule is only determined by the train's time to traverse interlocking routes and by constraints on departure times.

If $u = \langle a, r_i \rangle$ is a route node of train a, and r_{i+1} is the route following r_i on the path of a, then we let $u + 1 = \langle a, r_{i+1} \rangle$. That is, $u + 1$ is defined for all $u \in \mathcal{N} \setminus \mathcal{N}_S$. We let $L_{\langle a,r \rangle}$ be the time it takes a to traverse r, so we have the following *traversal time constraint*

$$t_{u+1} - t_u \geq L_u \qquad\qquad u \in \mathcal{N} \setminus \mathcal{N}_S. \tag{2}$$

Trains typically follow a public timetable and cannot depart from a station before the officially scheduled time. Furthermore, we need to specify when trains enter the dispatching area being modeled. At the beginning of a train's path or at a station platform, we limit the train's earliest departure time, which is the earliest the train may enter the following interlocking route. We let $\Gamma : \mathcal{N} \to \mathbb{R}$, where $\Gamma_u = \Gamma(u)$ is the *earliest release time* of u relative to t_o, and get the *release time constraint*

$$t_u - t_o \geq \Gamma_u \qquad\qquad u \in \mathcal{N} \tag{3}$$

If the train is subject to a *no-wait* condition, the inequalities (2) and (3) may become equalities. E.g., if the train is not allowed to stop at a certain signal or the time a train enters the controlled area is fixed. If there is no earliest release time for u, we make the constraint (3) redundant by setting $\Gamma_u = 0$; by (1), we already have $t_u \geq t_o$.

3.3. Timetable and Objective

A standard way to assess the quality of a schedule $\mathbf{t}$ is by comparing it to the published timetable T. The timetable will specify target arrival and departure times at specific points in the network, called *timing points*. From the point of view of dispatchers, these points are normally the home signals and exit signals of stations with scheduled stops. The times at these points are denoted as, respectively, *arrival time* (at the station) and *departure time* (from the station). The quality of schedule $\mathbf{t}$ is measured by a cost function $c(\mathbf{t}, T)$, which typically penalizes delays of trains at their timing points. For passenger trains, the cost function may only penalize delays at arrival since these are the ones that most influence passenger utility; a delayed departure is not a problem if the train catches up by the next station.

More formally, we let $\mathcal{N}_T$ be a set of route nodes designated as *timing nodes*. A *timetable* is a function $T : \mathcal{N}_T \to \mathbb{R}$. We let $T_u = T(u)$, so that $T_{\langle a,r \rangle}$ is the *target time* (or *target entry time*) of train a in route r (or at $\langle a, r \rangle$). We define the *delay* at each timing node $u \in \mathcal{N}_T$ as $t_u - T_u$ if $t_u > T_u$, and 0 otherwise, and introduce the *delay variable* η_u, with

$$\eta_u = \max(0, t_u - T_u) \qquad\qquad u \in \mathcal{N}_T \tag{4}$$

Note that, depending on the route r, the target (entry) time may be an arrival or departure time. If T_u is the target entry (resp. arrival, departure) time at u, then t_u is the scheduled entry (resp. arrival, departure) time at u and η_u is the delay in entry (resp. arrival, departure) time at u.

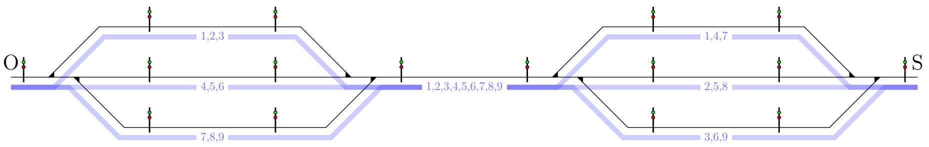


Figure 6: There are 9 possible paths from O to S. For a train travelling through a sequence of stations with multiple paths through each station, the total number of paths is, in the worst case, exponential in the number of stations.

3.4. Potential Conflicts and Selection Constraints

Until now, all our discussions have been about the free running of trains. In order to take the interaction between trains into account, we must now consider all potential scheduling conflicts between trains. Figure 3 shows an example of two trains crossing paths.

As described in Section 2, a potential conflict exists when two trains a, b make use of two incompatible (e.g., overlapping) interlocking routes r, q, respectively. In this case, we say a potential conflict exists between route notes $\langle a, r\rangle$ and $\langle b, q\rangle$. In Figure 3, $\langle a, AD\rangle$ and $\langle b, BD\rangle$ are in potential conflict, and so are $\langle a, DG\rangle$ and $\langle b, DF\rangle$.

Conflicts cannot occur in an actual schedule, and so we must decide which train goes first. If a goes first, then b can enter q only after a has left r. Vice versa, if b goes first, then a can enter r only after b has left q. This disjunctive precedence condition translates into a disjunctive constraint on the schedule of suitable route nodes on the paths of a and b.

We let $\mathcal{K}$ be the set of pairs of route nodes in potential conflict. Then, the following disjunctive constraint must be satisfied by every feasible schedule:

$$\begin{array}{c} t_v - t_{u+1} \geq \delta_u \\ \text{or} \\ t_u - t_{v+1} \geq \delta_v \end{array} \qquad \{u, v\} \in \mathcal{K} \tag{5}$$

where δ_u for $u = \langle a, r\rangle$ is the time it takes the length of train a to pass the signal at the end of interlocking route r, thus clearing the way for the next train.

4. Path Selection

In this section, we show how to extend our model to consider the existence of alternative paths for a train through the station. Different paths may exist from the entry point to the platform track and from the platform track to the exit point. Even the choice of platform (or platform track) may not be fixed in advance, although the official timetable may indicate a preferred platform (or platform track). Each path is a sequence of interlocking routes, as pictured in an example with two stations in Figure 6. As the figure shows, the number of possible paths can grow exponentially with the number of locations where multiple routing options are available.

Figure 7 shows the station from Figure 4 with some of the possible paths drawn in. Paths 5–9 are the paths shown in Figure 4. The paths entering from signal A and exiting through

Path name	1	2	3	4	5	6	7	8	9
Signals	ACFI	ADFI	ADGI	ADHI	BDFJ	BDGJ	BEFJ	BEGJ	BEHJ

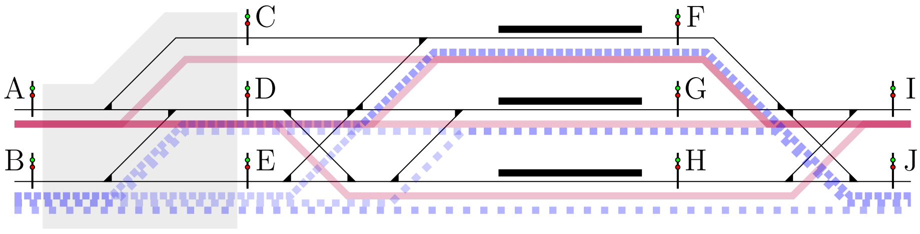


Figure 7: A collection of paths through the station in Figure 2. Each path is made up of three interlocking routes, identified by the adjacent pairs of signals in the path definition. The interlocking routes are uniquely defined by the names of their end signals, with the exception of DG and EG, which we define as they are drawn in the figure.

signal I are shown as solid, purple lines. The paths entering from signal B and exiting through signal J are shown as dashed, blue lines. We note that the interlocking routes DG and EG are not uniquely defined by their end signals. We define DG and EG as they are drawn in Figure 7. The alternative interlocking routes (with the detour to the lower track for DG and the earlier change to the middle track for EG) could be added under different names, which would increase the number of possible paths.

We now extend our model to allow for path selection. As before, we let $\mathcal{A}$ be the set of trains, and for train a, we let $\mathcal{N}(a)$ be the set of route nodes for a and $\mathcal{P}(a)$ be the set of paths available to a. Any path $p \in \mathcal{P}(a)$ is an ordered sequence of route nodes in $\mathcal{N}(a)$. If u is a route node of p, we denote by u_p^+ the route node which follows u on p (if it exists). For a node $u \in \mathcal{N}(a)$, we let $S(u) \subseteq \mathcal{N}(a)$ be the set of potential successors of u. That is, if $v \in S(u)$ then there is at least one path $p \in \mathcal{P}(a)$ such that $v = u_p^+$. For a node $u = \langle a, r \rangle \in \mathcal{N}(a)$, we denote by $\rho(u)$ the set of paths in $\mathcal{P}(a)$ which goes through (i.e., uses or contains) interlocking route r. Now, let $\mathcal{P} = \bigcup_{a \in \mathcal{A}} \mathcal{P}(a)$. We define the *path variable* $w_p \in \{0,1\}$ for $p \in \mathcal{P}$, which is 1 if and only if path p is selected. Since each train must be assigned exactly one path through the modelled area, we get the *path selection constraint*

$$\sum_{p \in \mathcal{P}(a)} w_p = 1 \qquad\qquad a \in \mathcal{A} \tag{6}$$

Next, for all route nodes $u \in \mathcal{N} \setminus \{o\}$ we introduce a variable $z_u \in \{0,1\}$ which is 1 if and only if (a path containing) u is selected. We get

$$z_u = \sum_{p \in \rho(u)} w_p \qquad\qquad u \in \mathcal{N} \setminus \{o\} \tag{7}$$

If $z_u = 1$ we say that node u is *active* (or selected).

Many of the constraints we have introduced in Section 3 now depend on the choice of path for each train. We generalize these constraints in the following sections.

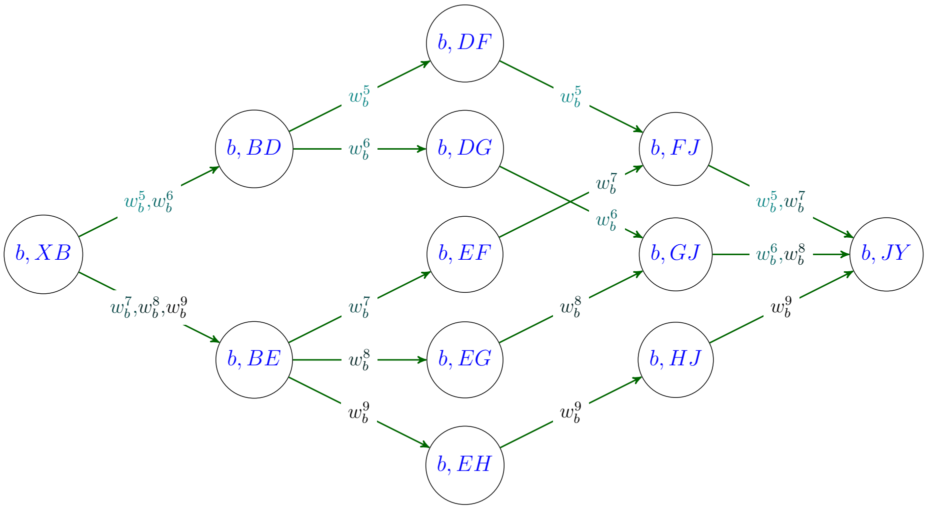


Figure 8: The route-node graph G_b of train b. Nodes correspond to the interlocking routes available for b, and an arc (u, v) means that the route associated with v starts at the signal where the route associated with u ends. The variables used to label each arc (u, v) represent the paths containing both u and v. The route node $\langle b, XB\rangle$ is the origin or the route node for the interlocking route leading to signal B, while the route node $\langle b, JY\rangle$ is a sink node (if the train leaves the modelled area) or the route node for the interlocking route leading away from signal J. In the figure, we are assuming that neither $\langle b, XB\rangle$ nor $\langle b, JY\rangle$ are path dependent. Any path $p \in \mathcal{P}(a)$ corresponds to a directed path from $\langle b, XB\rangle$ to $\langle b, JY\rangle$ in G_b. However, not all paths from $\langle b, XB\rangle$ to $\langle b, JY\rangle$ in G_b need to belong to $\mathcal{P}(a)$.

4.1. Path-Dependent Free Running Constraints

The following set of inequalities is a generalization of (2) to the case with path selection:

$$t_{u_p^+} - t_u - L_u \geq M\,(w_p - 1) \qquad u \in \mathcal{N} \setminus \mathcal{N}_S, p \in \rho(u) \tag{8}$$

where M is a suitable large positive constant. When $w_p = 1$, then (8) reduces to (2), otherwise it becomes redundant (this is the *big-M trick*). It is well known that big-M constraints are weak, in the sense that they do not help improve the value of the linear relaxation of the MILP formulation [20].

In a station with parallel platform tracks (as in Oslo Central Station), one can show that the system of inequalities (8) is dominated by the following family:

$$t_v - t_u \geq L_u \sum_{p \in \rho(u) \cap \rho(v)} w_p \qquad u \in \mathcal{N} \setminus (\mathcal{N}_S \cup \{o\}), v \in S(u) \tag{9}$$

where L_u, when $u = \langle a, r\rangle$, is the time for train a to run through r. Note that when $w_p = 1$ for some p containing both u and v, then (9) reduces to (2). Otherwise, when either or both route nodes are inactive, the constrain reduces to $t_v \geq t_u$ (where $v \in S(u)$).

Similarly, we get path dependent release-time constraints

$$t_u - t_o - \Gamma_u \geq M(z_u - 1) \qquad u \in \mathcal{N}_R \tag{10}$$

Again, one can show that this constraint is dominated by the following system of inequalities:

$$t_u - t_o \geq \Gamma_u z_u \qquad u \in \mathcal{N}_R \tag{11}$$

which does not involve the big-M constant. Indeed, if $z_u = 1$, then (11) reduces to (10). If $z_u = 0$, then (11) reduces to (1).

Figure 8 shows the route-node graph G_b for the station in Figure 7. We have omitted the arc weights, and have instead labeled each arc with the path variables on which it depends, in accordance with (8) (or (9)).

4.2. Path-Dependent Objective Functions

This subsection shows how to assess the cost of a schedule (and path selection) when trains can follow different routes in the station. There are two major considerations. First, the timing points may depend on the path. Second, some paths may be preferred to others, e.g., when the official timetable establishes the (preferred) stopping platform.

Path-dependent timing nodes. Let $\mathcal{N}_T(a) \subseteq \mathcal{N}_T$ be the timing nodes for train a. Then we rewrite (4) as follows

$$\begin{array}{lll} \eta_u = max(0, t_u - T_u) & \text{if } z_u = 1 & \\ \eta_u = 0 & \text{otherwise} & \end{array} \qquad u \in \mathcal{N}_T(a), a \in \mathcal{A} \tag{12}$$

We let $k_a \geq 0$ be the cost of 1 unit of delay of train a.

Path-dependent costs. To account for this cost component, we let $c_p \geq 0$ be the cost of choosing path $p \in P$.

Path-dependent objective function. The overall path-dependent objective function can be written as:

$$\min \sum_{a \in \mathcal{A}} \sum_{p \in \mathcal{P}(a)} c_p w_p + \sum_{a \in \mathcal{A}} \sum_{u \in \mathcal{N}_T(a)} k_a \eta_u \tag{13}$$

In order to express (16) using linear constraints, we can use the big-M trick. This results in the following family of inequalities:

$$\begin{array}{l} \eta_u \geq t_u - T_u - M(1 - z_u) \\ \eta_u \geq 0 \end{array} \qquad a \in \mathcal{A}, u \in \mathcal{N}_A^a \tag{14}$$

If we have $z_u = 1$, then $\eta_u \geq t_u - T_u$ and $\eta_u \geq 0$ hold together, and then the positive coefficient k_a in objective function will push the optimal value η_u^* down to $\max(0, t_u - T_u)$. When $z_u = 0$, only $\eta_u \geq 0$ holds (the other inequality becomes redundant), and we get $\eta_u^* = 0$.

Partitioning of path-dependent timing nodes. We now consider the case where $\mathcal{N}_T(a)$, the timing nodes for train a, can be partitioned into $\{\mathcal{N}_T^1(a), \ldots, \mathcal{N}_T^{r^a}(a)\}$, such that

$$\begin{aligned} \sum_{u \in \mathcal{N}_T^i(a)} z_u &= 1 & & & i \in \{1, \ldots, r^a\} \\ T_u &= T_v & \text{for } u, v \in \mathcal{N}_T^i(a) & & \end{aligned} \tag{15}$$

That is, train a will use precisely one timing node in $\mathcal{N}_T^i(a)$, and all timing nodes in $\mathcal{N}_T^i(a)$ have the same time in the published timetable.

For passenger trains with multiple path options in a station, the partitioning condition (15) holds if arrival and departure times are independent of path selection. This is usually the case, and we can generally assume that the above partition exists. When it does, we introduce only one delay variable per set in the partition, namely $\eta_a^i, \ldots, \eta_a^{r^a}$.

Finally, using (15) and the assumptions behind (9), one can show that the constraints (14) can be replaced by the family of constraints

$$\begin{aligned} \eta_a^i &\geq t_u - T_u \\ \eta_a^i &\geq 0 \end{aligned} \qquad a \in \mathcal{A}, i = 1, \ldots, r^a, u \in \mathcal{N}_T^i(a) \tag{16}$$

which does not contain the big-M constant. We let k_a^i be the cost of delaying train a by one unit at the nodes in $\mathcal{N}_T^i(a)$ (e.g., the parallel timing points this represents), and get

$$\min \sum_{a \in \mathcal{A}} \sum_{p \in \mathcal{P}(a)} c_p w_p + \sum_{a \in \mathcal{A}} \sum_{i=1}^{r^a} k_a^i \eta_a^i \tag{17}$$

4.3. Path-Dependent Selection Variables and Disjunctive Constraints

When we introduce alternative paths, potential conflicts become path-dependent, as they depend on whether certain tracks are used by certain trains.

More specifically, let $u = \langle a, r \rangle$ and $v = \langle b, s \rangle$ be two distinct route nodes, with $\{u, v\} \in \mathcal{K}$, where now $\mathcal{K}$ contains all pairs of route nodes in potential conflict, independently of whether or not the nodes are actually used by the trains. Then the potential conflict exists if and only if a path $p \in \rho(u)$ trough u for train a and a path $q \in \rho(v)$ through v for train b are selected, namely if node u and v are both active. In this case, we need to decide whether a precedes b or b precedes a in the contested track resource. The constraint (5) is extended as follows:

$$\begin{array}{c} t_v - t_{u_p^+} \geq \delta_u \\ \text{or} \\ t_u - t_{v_q^+} \geq \delta_v \end{array} \qquad \text{if } z_u = z_v = 1 \qquad \{u, v\} \in \mathcal{K} \tag{18}$$

To linearize the above disjunctive constraint we introduce, for $\{u, v\} \in \mathcal{K}$, binary selection variables $y_{uv}, y_{vu} \in \{0, 1\}$ which, as in (5), decide which of the two terms of the disjunction must be satisfied by the schedule. In particular, if $y_{uv} = 1$, then u precedes v and the first term is the valid one; if $y_{vu} = 1$, then v precedes u and the second term is the valid one. Since (18) only holds if both u and v are active, we have for all $\{u, v\} \in \mathcal{K}$

$$y_{uv} \le z_u, \quad y_{vu} \le z_u, \quad y_{uv} \le z_v, \quad y_{vu} \le z_v. \tag{19}$$

In any case, for all $\{u, v\} \in \mathcal{K}$ at most one selection variable can be one:

$$y_{uv} + y_{vu} \le 1 \tag{20}$$

Finally, for all $\{u, v\} \in \mathcal{K}$, when both u and v are selected, one selection variable must be one:

$$y_{uv} + y_{vu} \ge z_u + z_v - 1. \tag{21}$$

Note that if $z_u = 0$ or $z_v = 0$, (21) is redundant as the y variables are non-negative.

We are now ready to write the linear version of constraint (18), by exploiting once again the big-M trick:

$$\begin{array}{llll} (i) & t_v - t_{u_p^+} - \delta_{u_p^+} \ge M\,(y_{uv} + w_p - 2) & p \in \rho(u) & \\ & & & \{u, v\} \in \mathcal{K} \\ (ii) & t_u - t_{v_q^+} - \delta_{v_q^+} \ge M\,(y_{vu} + w_q - 2) & q \in \rho(v) & \end{array} \tag{22}$$

Let $u = \langle a, r \rangle$ and $v = \langle b, s \rangle$, and suppose $w_p = y_{uv} = 1$. It follows from (19) that, since $y_{uv} = 1$, both u and v are selected and train a precedes b. $w_p = 1$ implies that path p is selected for train a and, therefore, u_p^+ is the node following u for a. Since $w_p = y_{uv} = 1$, the r.h.s. of (22.i) is 0 and the constraint is active. On the other hand, since $y_{uv} = 1$, then $y_{vu} = 0$, and, with M suitably large, (22.ii) becomes redundant. A similar argument applies with the first and second term in (22) interchanged, when $y_{uv} = 0$ and $y_{vu} = w_q = 1$.

These path-dependent disjunctive constraints can be represented and visualized by the *path-dependent disjunctive graph* of in Figure 9, associated with two trains a, b. This graph contains as subgraphs the route-node graphs G_a, G_b of trains a and b, respectively, plus the origin, which is connected to the entry node of each train in the corresponding route-node graph. The green arcs belong to the train-specific route-node graphs, and they can be associated with the path variables as in Figure 8. Each arc with one endpoint in one route-node graph, and the other endpoint in the other route-node graph, is associated with one of the terms in a disjunction. A term becomes active if both the corresponding y variable and w variable are 1. Note that the graph may contain parallel arcs. To simplify notation, we let $\alpha = \langle a, AD \rangle$ and $\beta = \langle b, BD \rangle$.

In Figure 9 the red arcs correspond to the following set of inequalities (which are an instance of (22)):

$$\begin{array}{lll} t_\beta \ge t_{\langle a,DF \rangle} + \delta_{\langle a,DF \rangle} & \text{if} & y_{\alpha\beta} = w_a^2 = 1 \\ t_\beta \ge t_{\langle a,DG \rangle} + \delta_{\langle a,DG \rangle} & \text{if} & y_{\alpha\beta} = w_a^3 = 1 \\ t_\beta \ge t_{\langle a,DH \rangle} + \delta_{\langle a,DH \rangle} & \text{if} & y_{\alpha\beta} = w_a^4 = 1 \\ t_\alpha \ge t_{\langle b,DF \rangle} + \delta_{\langle b,DF \rangle} & \text{if} & y_{\beta\alpha} = w_b^5 = 1 \\ t_\alpha \ge t_{\langle b,DG \rangle} + \delta_{\langle b,DG \rangle} & \text{if} & y_{\beta\alpha} = w_b^6 = 1 \end{array}$$

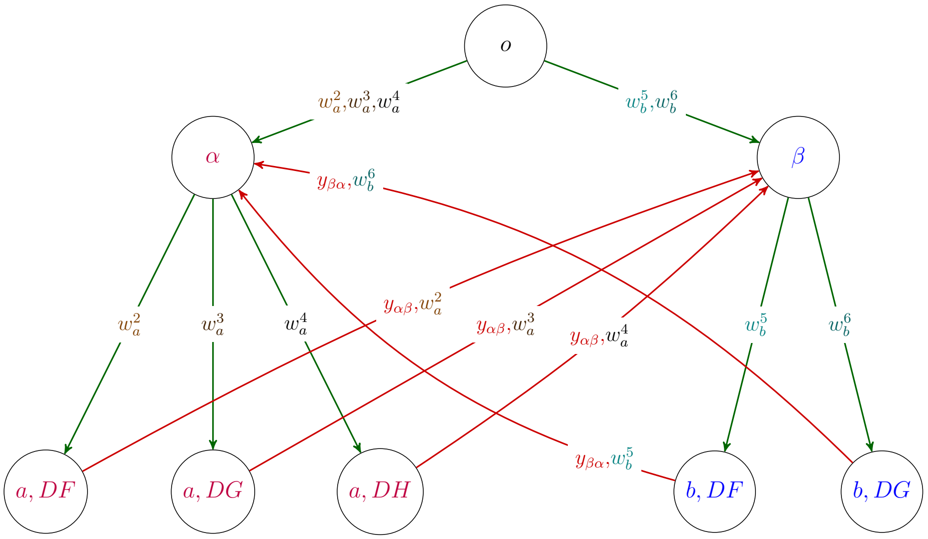


Figure 9: A path dependent disjunctive graph. Each red arc represents one of the two terms in a disjunctive constraint (22). One such arc is thus associated with a selection variable y and a path variable w. The term becomes active, i.e. the arc is chosen, when both the associated y and w variables are 1.

5. Solution Approach

Using the linearized constraints (8), (10), (14), and (22), we get the following MILP formulation for ODP. A listing of notation can be found in Table 1.

Symbol	Description
$\mathcal{A}$	Set of all trains
$\mathcal{N}$	Set of all route nodes
$\mathcal{N}(a)$	Set of all route nodes for train a
$\mathcal{N}_T$	Set of all timing nodes
$\mathcal{N}_T(a)$	Set of all timing nodes for train a
$\mathcal{N}_T^i(a)$	Element i of partition of timing nodes for train a
$\mathcal{N}_S$	Set of all sink nodes
o	Origin route node
u_p^+	Route node following u on path p
$\mathcal{P}$	Set of all paths
$\mathcal{P}(a)$	Set of all paths for train a
$\rho(u)$	Set of all paths containing route node u
$\mathcal{K}$	Set of all (potential) conflicts
$\mathbf{T}, T_u$	Timetable
$\mathbf{t}, t_u$	Schedule (variable)
η_u, η_a^i	Delay variable
y_{uv}	(precedence) selection variable
w_p	(path) selection variable
z_u	(route node) selection variable
k_a, k_a^i	Cost of unit delay to train a (in i)
c_p	Cost of selecting path p
$L_{\langle a,r\rangle}$	Time for train a to pass through interlocking route r
$\delta_{\langle a,r\rangle}$	Time for the length of train a to pass the signal at the end of r

Table 1: Listing of notation

$$
\begin{array}{lllll}
\textbf{min} & & \sum_{a\in\mathcal{A}}\sum_{p\in\mathcal{P}(a)} c_p w_p + \sum_{a\in\mathcal{A}}\sum_{u\in\mathcal{N}_T(a)} k_a \eta_u & & \\
\textbf{s.t.} & (ia) & \sum_{p\in\mathcal{P}(a)} w_p = 1 & a\in\mathcal{A} & \\
\textbf{s.t.} & (ib) & \sum_{p\in\rho(u)} w_p = z_u & u\in\mathcal{N}\setminus\{o\} & \\
 & (iia) & y_{uv}+y_{vu} \geq \sum_{p\in\rho(u)} w_p + \sum_{q\in\rho(v)} w_q - 1 & \{u,v\}\in\mathcal{K} & \\
 & (iib) & y_{uv}+y_{vu} \leq 1 & \{u,v\}\in\mathcal{K} & \\
 & (iic) & y_{uv} \leq z_u & \{u,v\}\in\mathcal{K} & \\
 & (iid) & y_{uv} \leq z_v & \{u,v\}\in\mathcal{K} & \\
 & (iii) & t_u - t_o - \Gamma_u \geq M\,(z_u - 1) & u\in\mathcal{N} & \\
 & (iv) & t_{u_p^+} - t_u - L_u \geq M\,(w_p - 1) & u\in\mathcal{N}\setminus\mathcal{N}_S, p\in\rho(u) & (23)\\
 & (va) & t_v - t_{u_p^+} - \delta_u \geq M\,(y_{uv} + w_p - 2) & \{u,v\}\in\mathcal{K}, p\in\rho(u) & \\
 & (vb) & t_u - t_{v_p^+} - \delta_v \geq M\,(y_{vu} + w_q - 2) & \{u,v\}\in\mathcal{K}, q\in\rho(v) & \\
 & (vi) & \eta_u - t_u + T_u \geq M\,(z_u - 1) & a\in\mathcal{A}, u\in\mathcal{N}(a) & \\
 & & t_u \geq 0 & u\in\mathcal{N} & \\
 & & \eta_u \geq 0 & a\in\mathcal{A}, u\in\mathcal{N}_T(a) & \\
 & & y_{uv}\in\{0,1\} & \{u,v\}\in\mathcal{K} & \\
 & & w_p\in\{0,1\} & p\in\mathcal{P} & \\
 & & z_u\in\{0,1\} & u\in\mathcal{N}\setminus\{o\} &
\end{array}
$$

In Section 4, we discussed how we can strengthen (23.iii), (23.iv), and (23.vi), replacing them with (9), (11), and (16), respectively.

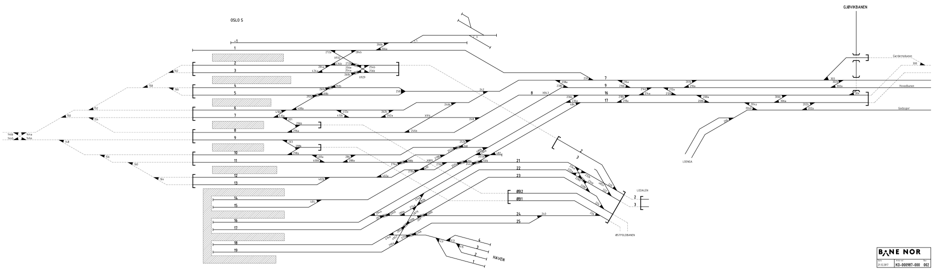
5.1. Delayed Variable and Constraint Generation

Already in a moderately sized instance of ODP, the number $|\mathcal{K}|$ of potential scheduling conflicts can grow prohibitively large. Therefore, rather than generating a full instance with all constraints (23.v) from the start, we prefer to solve a sequence of smaller instances by applying the delayed constraint generation approach. The idea is to start solving to optimality a model with much fewer constraints. We then check if any of the missing constraints are violated by the current optimal solution. If this is not the case, then the solution can be shown to be optimal for the full problem (with all constraints). Otherwise, the violated constraints are added to the model, and the process is iterated.

We start with a model $\mathcal{M}^0$, which is (23) with all constraints of type (23.v) removed, and without any y-variables. Then, we use the algorithm outlined as follows.

1. Set $i \leftarrow 0$.

2. Find the optimal solution t^i, w^i, y^i, μ^i to $\mathcal{M}^i$.



Source: Bane NOR (CC BY-SA 4.0)

Figure 10: Schematic track layout of Oslo Central Station. The station is laid out approximately east-to-west, so this schematic is oriented north-up.

Table 2: Comparison between different MILP formulations for ODP using different versions of the constraints (23.iii), (23.iv), and (23.vi). For comparison, a set of 200 instances (220 trains over 4 hours with random delays) have been solved using each formulation, and the values shown in this table are the averages for each formulation. The fastest formulation has been highlighted.

(23.iii),(23.iv)	(23.vi)	t_{total}	t_{last}	t_{sep}	MIP	Nodes	Vars	Constr
big-M	big-M	13.63	1.00	0.25	17.07	21778.87	1035.29	517.64
big-M	modified	16.47	0.66	0.36	25.45	26405.58	1300.69	650.35
big-M	combined	15.11	0.63	0.34	25.43	22796.97	1273.94	636.97
modified	big-M	14.83	1.18	0.25	16.88	29430.74	1001.78	500.89
modified	modified	29.30	1.83	0.31	21.12	49842.10	1147.05	573.52
modified	combined	24.17	1.53	0.27	19.64	40426.82	1125.34	562.67
combined	**big-M**	**12.73**	**0.94**	**0.24**	**16.66**	**20313.77**	**1036.53**	**518.26**
combined	modified	23.07	1.57	0.28	18.55	34165.39	1136.41	568.21
combined	combined	16.98	1.22	0.26	17.59	25044.31	1093.82	546.91

(a) if there exist a potential conflict pair $\{u, v\} \in \mathcal{K}$ such that paths $p \in \rho(u)$, $q \in \rho(v)$ are chosen (i.e. $w_p^i = w_q^i = 1$) and constraint (18) is violated, create $\mathcal{M}^{i+1}$ by adding to the model the associated constraints (19), (20), (21) and (22), and update $i \leftarrow i + 1$. Go to 2.

(b) else the solution is optimal for (23).

Checking for violated inequalities can be done very efficiently. In Appendix A we give more details on our constraint generation (conflict detection) algorithm.

6. Computational Results

To test our approach, we find the optimal dispatching solution in a variety of traffic instances in Oslo Central Station, shown in Figure 10. The station has 19 platform tracks, 1 west-bound line, 4 east-bound lines, and 2 east-bound exits to technical areas. Oslo Central Station has 19 tracks. Track 1 and tracks 14–19 are east-bound only, while tracks 2–13

Table 3: Computational results for Oslo Central Station. Each instance is 6 hours of rush-hour traffic, with a total of 330 trains. N_d is the number of trains that has been delayed to create the random instance, and each delayed train was delayed by a random number of minutes between 1 and 15.

N_d	t_{total}	t_{last}	t_{sep}	MIP	Nodes	Vars	Constr
0	31.75	2.19	0.39	19	27842	1576	788
1	44.83	1.89	0.48	24	39388	1628	814
2	15.46	1.52	0.27	13	18871	1486	743
3	41.33	1.41	0.50	24	27607	1662	831
4	28.17	1.01	0.34	16	43545	1586	793
5	32.61	2.65	0.39	20	27394	1572	786
6	29.99	2.77	0.36	17	31249	1562	781
7	56.48	2.18	0.62	30	41290	1736	868
8	41.42	2.23	0.47	23	29717	1652	826
9	27.63	1.77	0.38	18	44345	1566	783
10	47.22	1.99	0.49	24	39800	1690	845
11	17.89	1.17	0.29	14	13660	1556	778
12	42.11	1.77	0.49	24	32576	1600	800
13	22.29	1.87	0.33	16	26874	1542	771
14	34.46	2.48	0.40	20	27559	1612	806
15	36.59	1.91	0.47	22	24433	1734	867
16	27.39	1.64	0.38	18	35316	1568	784
17	27.00	2.01	0.38	18	21565	1564	782
18	31.92	2.87	0.40	19	28043	1618	809
19	21.70	1.80	0.35	16	13584	1540	770
20	31.87	2.14	0.39	18	29370	1580	790
21	25.43	2.68	0.35	16	36534	1574	787
22	18.38	2.18	0.28	14	11386	1574	787
23	23.60	2.59	0.36	17	28187	1610	805
24	28.07	2.19	0.34	16	19580	1642	821
25	29.39	2.43	0.44	21	34381	1604	802
26	32.68	1.43	0.40	19	21195	1592	796
27	48.98	2.42	0.48	25	45070	1644	822
28	18.66	1.71	0.28	13	18206	1604	802
29	27.29	1.97	0.38	18	36655	1576	788
30	24.44	2.06	0.33	16	17919	1568	784

Table 4: Computational results for Oslo Central Station. Each instance is 24 hours of traffic, with two rush-hour periods of 4 hours each and a total of 996 trains. N_d is the number of trains that has been delayed to create the random instance, and each delayed train was delayed by a random number of minutes between 1 and 15.

N_d	t_{total}	t_{last}	t_{sep}	MIP	Nodes	Vars	Constr
0	43.45	3.02	1.64	19	19721	3064	1532
1	47.47	2.99	1.81	22	34519	3016	1508
2	89.21	3.48	2.69	32	51796	3202	1601
3	63.15	2.34	2.08	25	31459	3166	1583
4	53.14	2.91	1.69	21	53566	3208	1604
5	54.38	3.16	1.80	21	32423	3200	1600
6	36.19	2.88	1.31	17	34178	3068	1534
7	57.66	3.71	1.89	24	28443	3176	1588
8	23.80	2.52	0.95	12	22428	3010	1505
9	36.60	2.73	1.32	16	23108	3100	1550
10	54.25	2.71	1.70	21	39828	3108	1554
11	91.07	2.48	2.75	34	54384	3246	1623
12	53.08	2.64	1.77	22	59062	3070	1535
13	35.81	3.29	1.36	17	26755	3086	1543
14	36.29	3.22	1.27	16	20789	3044	1522
15	55.57	3.87	1.84	23	32907	3114	1557
16	76.93	3.95	2.04	25	34058	3202	1601
17	50.44	2.75	1.62	20	61812	3066	1533
18	31.90	2.71	1.30	16	30493	3004	1502
19	67.57	3.44	2.09	26	35096	3108	1554
20	26.92	2.54	1.13	14	26348	2924	1462
21	57.52	3.48	1.75	21	26297	3224	1612
22	65.72	3.19	1.95	25	46235	3130	1565
23	61.08	3.70	2.08	26	54550	3182	1591
24	56.80	3.05	1.74	22	18673	3166	1583
25	33.78	2.33	1.30	17	18268	3100	1550
26	71.83	3.03	2.35	30	40423	3078	1539
27	47.63	2.66	1.73	21	46208	3024	1512
28	37.13	3.35	1.38	17	32929	3130	1565
29	84.11	1.86	2.46	32	49202	3132	1566
30	51.30	3.97	1.80	21	29310	3134	1567

are both east-bound and west-bound. Tracks 2–8 are primarily used for west-bound traffic, while tracks 9–13 are primarily used for east-bound traffic. To the west, all tracks collect into a west-bound double-track tunnel. This tunnel is the main line connecting the east and the west of Norway by rail and is very busy. To the east, traffic can go north-east on Brynsbakken to one of the three lines Gjøvikbanen, Hovedbanen, and Romeriksporten (Gardermobanen); southeast to Østfoldbanen or the depot at Lodalen; or south to the yard at Haven. To the east, tunnels allow traffic to move at different levels in order to improve traffic flow. The tracks are divided into hundreds of track circuits, allowing effective use of sectional release (see Appendix A). In order to construct our experiments, we have been given insight into restricted-access documents detailing the infrastructure of Oslo Central Station and the surrounding area. In particular, we have been able to use the real track-circuit and signal layout, though we have had to estimate the exact sizes of each track circuit. In order to generate the timetable, we have used the public listing of arrivals and departures at the station. With permission from Bane NOR, we have published the infrastructure data for Oslo Central Station in a companion paper [14].

All our instances are based on repeating the traffic scheduled between 4 p.m. and 5 p.m. on a weekday, during the height of the after-work commute out of Oslo city center. We test our approach under an extra heavy load by repeating this very busy hour instead of following the published timetable. The number of trains scheduled to move through the station this hour is 55, with some trains arriving from or departing to the neighboring depot. For traffic outside of rush hour (for longer simulations), we have selected 38 of the rush-hour trains. Whenever a track connection existed, the trains were allowed to use both their scheduled track and the other track on the same platform. As possible paths, we allowed the paths trains are observed to take to their designated track and paths suggested as alternatives by dispatchers at Bane NOR. We ran our experiments on a MacBook Pro (15-inch, 2016) with a 2.9 GHz Quad-Core Intel Core i7 processor and 16 GB 2133 MHz LPDDR3 memory, running macOS 10.15, Python 3.8.2, and Gurobi 9.0.3 with default settings.

In order to compare the different formulations we present in Section 5, we have generated a set of 200 instances, each consisting of 4 consecutive hours of rush-hour traffic. In each instance, we randomly select trains and give them a random delay between 1 and 15 minutes. The number of trains selected for delay is between 1 and 30. We then solve all 200 instances using each of the formulations we compare.

The *big-M* formulation is the one given in (23). In rows labeled *modified*, the corresponding constraints have been replaced by their strengthened versions, described immediately following (23). In rows labeled *combined*, the corresponding constraints have been supplemented by their strengthened versions. Table 2 shows the average performance statistics. The columns of the table are (t_{total}) the total processing time, (t_{last}) the processing time of the last MIP to be solved, (t_{sep}) the time spent separating conflict constraints, (MIP) the number of MIP models solved, (Nodes) the total number of MIP nodes processed, (Vars) the number of selection variables generated, and (Constr) the number of selection constraints generated.

Table 2 shows a surprising result: replacing (23.iii) and (23.iv) with stronger, non-big-M

versions slows down the solution process. Comparing the first three rows of Table 2 with the middle three rows, we see that the number of MIPs solved is smaller when using the strengthened constraints. This indicates that the intermediate solutions, and therefore the conflict detection process, might be better. However, the number of MIP nodes processed is much larger, indicating that the branch-and-bound process is less effective. When we replace (23.vi), we get even slower running times. In this case, strengthening the inequality removes the explicit connection between the path variables and the delay variables, which could explain the increased number of MIP nodes processed. The highlighted row of Table 2 shows the formulation with the best performance in our comparison experiments. In this formulation we have supplemented (23.iii) and (23.iv) with their strengthened version, and kept (23.vi). This results in the number of MIP solves and the number of MIP nodes processed being comparatively low. We use this formulation in the rest of our experiments.

Table 3 shows performance results from an experiment very similar to the comparison experiment, but with six consecutive hours of traffic. The new column (N_d) is the number of trains delayed to create the instance. Each instance has 330 trains moving through the station. Most trains have options for which path to take to their designated platform, and some trains have options for which platform to take. In each instance, the total number of path selection variables is 798. All the instances are solved in under 1 minute. This is fast enough that dispatchers can use the suggested solutions; the traffic pattern does not change too much in 1 minute. Furthermore, once a train is approaching the station, the signals will have been set, and changing the train's schedule will be very work-intensive. Therefore, there is no great need for a faster refresh rate than every minute; dispatchers need time to implement the planned schedule.

Table 4 shows the performance results from an experiment where each instance is made up of 24 hours of traffic: two 4-hour periods of rush-hour traffic and three periods of reduced traffic. Each instance has 996 trains moving through the station, and the total number of path selection variables is 2520. All instances are solved within 100 seconds. Similar to the above situation, this is fast enough to be useful in real-time applications when dispatchers and schedulers plan this far ahead.

7. Conclusions and Future Work

The computational results reported in the previous section show that our approach can solve the ODP in a large passenger train station up to a 24-hour planning horizon in a reasonable amount of time. As a part of the innovation project GOTO with the Norwegian infrastructure manager Bane NOR, a prototype based on our approach will undergo a field-test campaign from the fall of 2021. The dispatchers at Oslo Central Station control center will be involved in the testing.

Even though the experiments on real-life data show that the approach will work in practice, the field campaign may present new challenges. Also, further research may be needed to tackle even larger stations with more congested traffic, which exist in other European railways.

In the GOTO project, we also developed a prototype, already under testing at Oslo Central Station, to dispatch trains on the lines incident to the station. The final objective of the GOTO project is to develop a decomposition approach to combine the two prototypes to control the traffic over the entire Greater Oslo region. The final prototype will be tested at Oslo Central Station in June 2022.

Based on discussions with dispatchers at Oslo Central Station, we have decided to rely on predetermined paths for the routing within the station. On the one hand, this approach aligns well with how dispatchers route trains in the station, making it easy for dispatchers to implement suggested solutions. On the other, using predetermined paths restricts the search space. In order to further improve our approach, we want to explore more options for path generation. First, we want to let dispatchers control which paths are available for each train, including drawing new paths. Second, we want to design an algorithm that generates paths on-demand based on the current traffic in the station.

References

[1] Ravindra K. Ahujia, Thomas L. Magnanti, and James B Orlin. Network flows: Theory, algorithms and applications. *New Jersey: Rentice-Hall*, 1993.

[2] Cynthia Barnhart, Christopher A Hane, and Pamela H Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48(2):318–326, 2000.

[3] Sveinung Berg Bentzrød. Bane Nor tør ikke love bedre punktlighet i 2020. *Aftenposten*, 2019.

[4] Valentina Cacchiani, Dennis Huisman, Martin Kidd, Leo Kroon, Paolo Toth, Lucas Veelenturf, and Joris Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63:15–37, 2014.

[5] Gabrio Caimi, Martin Fuchsberger, Marco Laumanns, and Marco Lüthi. A model predictive control approach for discrete-time rescheduling in complex central railway station areas. *Comput. Oper. Res.*, 39(11):2578–2593, 2012.

[6] Francesco Corman and Lingyun Meng. A review of online dynamic models and algorithms for railway traffic management. *IEEE Trans. Intell. Transp. Syst.*, 16(3):1274–1284, 2015.

[7] Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.

[8] Wei Fang, Shengxiang Yang, and Xin Yao. A survey on problem models and solution approaches to rescheduling in railway networks. *IEEE transactions on intelligent transportation systems*, 16(6):2997–3016, 2015.

[9] S. Foglietta, G. Leo, C. Mannino, P. Perticaroli, and M. Piacentini. An optimized, automatic tms in operations in roma tiburtina and monfalcone stations. *WIT Transactions on the Built Environment*, 135:635–647, 2014.

[10] Steven Harrod. Modeling network transition constraints with hypergraphs. *Transportation Science*, 45(1):81–97, 2011.

[11] Bisheng He, Peng Chen, Andrea D'Ariano, Lufeng Chen, Hongxiang Zhang, and Gongyuan Lu. A microscopic agent-based simulation for real-time dispatching problem of a busy railway passenger station. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7, 2020.

[12] Leonardo Lamorgese and Carlo Mannino. A Noncompact Formulation for Job-Shop Scheduling Problems in Traffic Management. *Operations Research*, 67(6):1586–1609, 2019.

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

[14] Andreas Nakkerud. Rail infrastructure data for oslo central station. *Data in Brief*, In Press.

[15] Paola Pellegrini, Grégory Marlière, and Joaquin Rodriguez. Optimal train routing and scheduling for managing traffic perturbations in complex junctions. *Transportation Research Part B: Methodological*, 59(C):58–80, 2014.

[16] Maurice Queyranne and Andreas S Schulz. *Polyhedral approaches to machine scheduling.* TU, Fachbereich 3, Berlin, 1994.

[17] Edwin Reynolds, Matthias Ehrgott, Stephen J. Maher, Anthony Patman, and Judith Y.T. Wang. A multicommodity flow model for rerouting and retiming trains in real-time to reduce reactionary delay in complex station areas. *Optimization Online*, 2020.

[18] SINTEF. GOTO Project. `https://www.sintef.no/en/projects/2019/goto/`, Accessed: 28 July 2021.

[19] Gregor Theeg and Sergej Vlasenko, editors. *Railway Signalling and Interlocking.* PMC Media House, Leverkusen, Germany, 3rd edition, 2020.

[20] Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

[21] Taomei Zhu, Jose Manuel Mera, Berta Suarez, and Joaquín Maroto. An agent-based support system for railway station dispatching. *Expert Syst. Appl.*, 61:39–52, 2016.

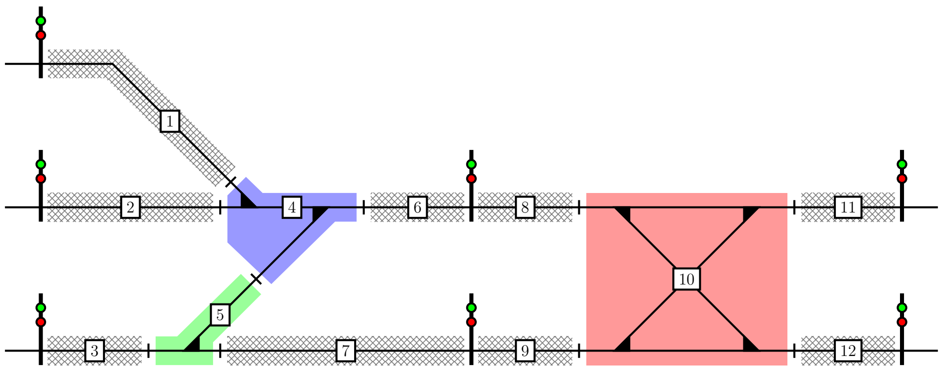


Figure A.11: The figure shows an area of tracks divided into 12 track circuits, shown as shaded or crosshatched areas. The track circuits are delimited by the signals and the small hatches drawn on the lines. The large, shaded track circuit on the right prevents the two parallel tracks going through it from being used simultaneously.

Appendix A. Detailed Conflict Modelling

In order to model Oslo Central Station (or any other large passenger station) with sufficient accuracy, it is necessary to understand the low-level business rules of operating the rail infrastructure that makes up the station. In this appendix, we explain the details behind our conflict detection approach. These are the details that allow us to achieve feasible schedules without adding safety margins.

Appendix A.1. Track Infrastructure

Up to this point, we have considered the track infrastructure to be divided into interlocking routes with potential conflicts between interlocking routes that share some physical resources. While the interlocking routes are the finest division from a scheduling point of view, these routes are further divided into *track circuits* for actual conflict detection. Figure A.11 shows an example of how tracks can be divided into track circuits.

Track circuits can be seen as atomic track elements since they partition the track infrastructure; no two track circuits may share any piece of track. Track circuits are equipped with detectors that can tell if a train occupies the track circuit. For the purpose of conflict detection, we regard an interlocking route as a sequence of track circuits $r = (c_1, \ldots, c_n)$. A pair of routes are in potential conflict when they share at least one track circuit.

In our discussion so far, if one train must wait for another, the preceding train must enter the following route in its path before the waiting train may proceed (Section 3.4). By looking at the underlying track circuits of an interlocking route, we may introduce alternative, less restrictive variants of the precedence constraints.

When a train enters an interlocking route, some of the track circuits following the upcoming signal are temporarily reserved as an added layer of protection against collisions.

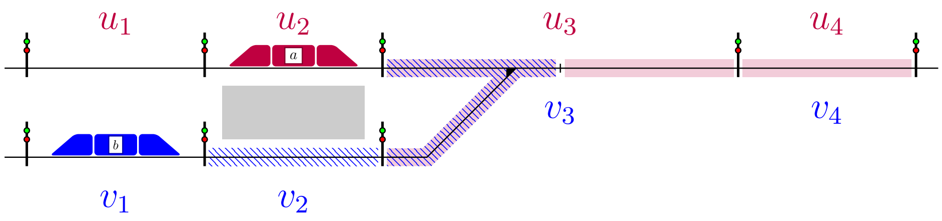


Figure A.12: If the blue train wants to enter v_2, it must reserve the blue, hatched track circuits, including the first track circuit of v_3 as a safety zone. This would prevent the purple train from entering u_3 until the safety zone is released. Conversely, if the purple train wants to enter u_3, it must reserve the purple, shaded track circuits, including u_4 as a safety zone, this would prevent the blue train from entering v_2 until the first track circuit of u_3 is cleared by the purple train and released.

Thus, two interlocking routes can be in potential conflict even if they do not share a track circuit directly.

Appendix A.2. Track Circuit Reservations

When a train enters an interlocking route, it must reserve all track circuits that are part of that route and possibly some track circuits forming a *safety zone* after the signal at the end of the route.

Any pair of trains reserving the same track circuit represent a potential scheduling conflict. In order to check if a candidate schedule realizes a potential conflict, we create an interval graph for each track circuit, where each interval represents a reservation of the track circuit. If any pair of intervals overlap, there is a realized scheduling conflict, and the candidate schedule is not feasible.

Track circuits reservations end and we say the track circuits are *released*, either after they have been passed by the train, when the train exits the containing interlocking route, or, in the case of safety zones, after a fixed amount of time.

Appendix A.2.1. Safety Zones

Signals offer the primary protection of trains moving in interlocking routes. No collision is ever possible as long as no train ever passes a signal at danger (red light). *Safety zones* (or interlocking route *overlaps* [19]) offer additional protection against collision if a train should pass a signal at danger accidentally.

Figure A.12 shows an example of a safety zone. Train b (blue) is about to enter route v_2, and train a (purple) is about to enter u_3. In order to protect against collision, train b must reserve not only v_2, but also the first track circuit of v_3. If b were to pass the signal protecting v_3 (and u_3), an automatic emergency break could stop b in the safety zone. Were it not for this safety zone, train a could occupy the points in u_3 while b enters v_2. Should b pass the following signal, it could crash into a before an emergency break could bring b to a stop.

Table A.5: When operating with safety zones, there are three types of conflicts between trains. First, two trains may want to use the same track circuit(s) as a safety zone. Second, a train may want to enter the safety zone of another train. And third, two trains may want to enter the same track circuit(s).

Purple	**Blue**	**Variables**	**Description**
Enter u_2	Enter v_2	$y_{u_2v_2}, y_{v_2u_2}$	Conflicting safety zones
Enter u_3	Enter v_2	$y_{u_3v_2}, y_{v_2u_3}$	Purple train enters safety zone of blue train
Enter u_2	Enter v_3	$y_{u_2v_3}, y_{v_3u_2}$	Blue train enters safety zone of purple train
Enter u_3	Enter v_3	$y_{u_3v_3}, y_{v_3u_3}$	Both trains enter the same track circuit(s)

In order to prevent a collision, a safety zone must contain enough track circuits to cover the minimum braking distance of the train approaching the corresponding signal. Unlike the track circuits making up an interlocking route, the track circuits of a safety zone may be released before they are used by the reserving train. Consider the example in Figure A.12 again. When b enters v_2, it is going to stop at the adjacent platform. Then, a will leave the platform and enter u_3. The blue, crosshatched track circuits are reserved by b, and the purple-shaded track circuits are the ones a must reserve. A certain amount of time after b enters v_2 we can say for certain that b must have slowed. Otherwise, it would have reached the following signal. At this time, b can release the track circuits making up its safety zone in v_3, and a can reserve the track circuits on its path. Track circuits in a safety zone are reserved for a fixed amount of time.

Appendix A.2.2. Route Release and Sectional Release

When a track circuit is part of an interlocking route, it is released either with the route or once the reserving train has passed through the track circuit itself.

We use Figure A.12 to illustrate the difference between route and sectional release. If we let train a take precedence over b, then b cannot enter v_2 before a releases the first track circuit of u_3, since this track circuit is the safety zone required by b when entering v_2. Under route release, a must fully enter u_4 before the contested track circuit is released, and b may proceed. Under sectional release, the track circuit can be released once a has passed fully into the last track circuit of u_3.

Under sectional release, train b may proceed earlier than under route release. Even if a must stop at the signal protecting u_4, a may proceed into v_2. In general, sectional release allows closer scheduling than route release.

Appendix A.3. Conflict Types

We let $u = \langle a, r\rangle$ and $v = \langle b, s\rangle$ be route nodes. There is a potential scheduling conflict between u and v when they have a track circuit in common, either as part of the route itself or a part of a safety zone. The associated precedence constraints depend on the type of reservations made by the route nodes. A pair of routes may cause multiple potential scheduling conflicts. We add constraints to make sure none of the potential conflicts are realized. The form of these constraints depends on what the yielding train is waiting for. Table A.5 and Figure A.13 show the different types of potential scheduling conflicts.

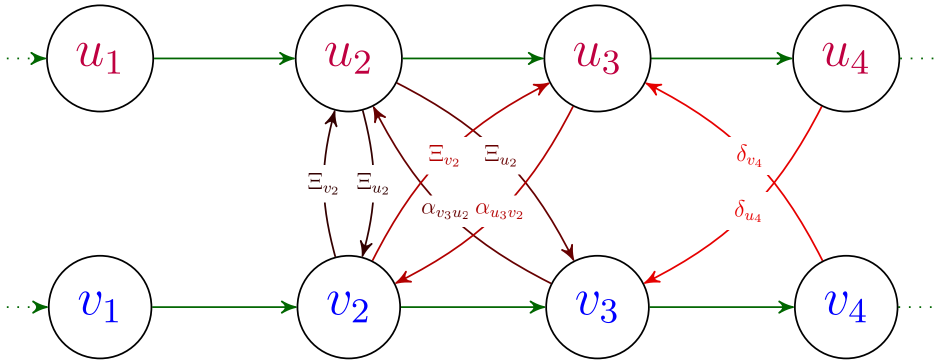


Figure A.13: Four disjunctive precedence constraints of three different types. The physical situation is illustrated in Figure A.12, and the conflicts are described in Table A.5.

Waiting for Route Release. When one train is waiting for a preceding train to fully clear its route, the waiting train is waiting for a *route release*. This is, for example, the case when both trains are going to the same signal. In Figure A.12, if a precedes b under route release, then b cannot enter v_2 before a has fully left u_3 (and fully entered u_4). We get the constraint

$$t_{v_2} - t_{u_4} \geq \delta_{u_4} \qquad \text{if} \qquad y_{u_3v_2} = 1 \tag{A.1}$$

where we note that the precedence decision is between u_3 and v_2, while it is u_4 that is used in the constraint. The arcs (u_4, v_3) and (v_4, u_3) in Figure A.13 both represent route release constraints.

Waiting for Sectional Release. If the route of the waiting train is crossing the route of the preceding train sufficiently far from the end of that route, then *sectional release* may apply. In sectional release, individual track circuits are released once they are passed. Since, when we reschedule, we may assume trains will only stop at signals, we can compute when a train passes a track circuit based on when it enters the route containing that track circuit. This will work as long as the track circuit is not part of the area where the train may stop before a signal. If train a precedes train b in Figure A.12, it is natural to apply sectional release.

We let α_{uv} be the time it takes for the preceding train (route node u) to get past the end of the last track circuit on the route of the waiting train (route node v). In the example in the figure we then get the constraint

$$t_{v_2} - t_{u_3} \geq \alpha_{u_3v_2} \qquad \text{if} \qquad y_{u_3v_2} = 1 \tag{A.2}$$

The arcs (u_3, v_2) and (v_3, u_2) in Figure A.13 both represent sectional release constraints.

Waiting for Safety Zone Release. Waiting for a safety zone to be released is much like waiting for a sectional release. In Figure A.12, if v_2 precedes u_3, then train a only has to wait a fixed time after b enters v_2, instead of waiting for b to enter v_3 (which in this case would cause a different potential conflict with a). We let Ξ_u be the amount of time the safety zone corresponding to u must be reserved. Then,

$$t_{v_2} - t_{u_3} \geq \Xi_{u_3} \qquad \text{if} \qquad y_{u_3 v_2} = 1 \tag{A.3}$$

The arcs (u_2, v_2), (v_2, u_2), (u_2, v_3), and (v_2, u_3) in Figure A.13 all represent waiting for a safety zone to be released.

Paper IV

Rail Infrastructure Data for Oslo Central Station

Andreas Nakkerud

Submitted to *Data in Brief.*