TU Darmstadt

Master Thesis

Metro Real-Time Disposition Optimization using the Alternative Graph Model

Author:
Florian Beutel

Supervisor:
Julian Harbarth

A thesis submitted in fulfillment of the requirements for the degree of M.Sc. Computer Science

in the

Algorithm Group
Department of Computer Science

July 9, 2023

Declaration of Authorship

TU DARMSTADT

Abstract

Department of Computer Science

M.Sc. Computer Science

Metro Real-Time Disposition Optimization using the Alternative Graph Model

by Florian BEUTEL

Real-time train disposition is an important optimization problem to avoid propagation of delays in railway networks. This thesis discusses the extension of an existing real-time train disposition model to incorporate the constraints and objective function of the 2022 RAS Problem Solving Competition. We extend the underlying alternative graph to model dynamic minimal run and headway times based on whether a train stops at a station or passes it. Furthermore, we include the stop or pass decisions as part of the alternative graph and allow for track changes as an additional timetable amendment. To solve the modified model, we present a branch-and-bound algorithm that optimizes for the multi-component target function of the 2022 RAS problem. Additionally, we introduce two heuristics to reduce the size of the alternative graph generated for a specific problem instance. We evaluate our approach using the infrastructure and timetable model of the London Elizabeth Line. The evaluation shows that the integration of track decisions is the main driver of the modified model's complexity.

TU DARMSTADT

Zusammenfassung

Fachbereich Informatik

M.Sc. Informatik

Echtzeit Metro Dispositionsoptimierung mittels des Alternativen-Graph-Modells

von Florian Beutel

Echtzeit Zug Disposition ist ein wichtiges Optimierungsproblem um das Ausbreiten von Verspätungen in Zugnetzen zu verhindern. Diese Thesis behandelt die Erweiterung eines existierenden Echtzeit Zug Dispositionsmodels um die Bedingungen und Zielfunktion des RAS Problemlösungswettbewerbs von 2022 zu integrieren. Wir erweitern den zugrundeliegende Alternativen-Graph sodass er dynamische minimale Fahr- und Taktzeiten abbilden kann. Die Dynamik beruht auf der Entscheidung ob ein Zug an einer Station hält oder den Halt ausfallen lässt. Des Weiteren modellieren wir diese Entscheidungen direkt im Alternativen-Graph und erlauben zusätzlich Gleiswechsel als weitere Fahrplananpassung. Um das modifizierte Model zu lösen, stellen wir einen angepassten Branch-and-Bound Algorithmus vor, der eine Zielfunktion mit mehreren Komponenten optimiert. Fie Zielfunktion ist die des RAS Problemlösungswettbewerbs von 2022. Darüber hinaus präsentieren wir zwei Heuristiken, die die Größe des Alternativen-Graphs der für eine spezifische Probleminstanz erstellt wird reduzieren. Wir evaluieren unseren Ansatz mittels des Infrastrukturmodels und Fahrplans der Elizabeth Line in London. Die Evaluation zeigt, dass die Einbindung der Gleiswechsel ein Hauptfaktor für die erhöhte Komplexität des erweiterten Models ist.

Contents

Declaration of Authorship **iii**

Abstract **v**

Zusammenfassung (German Abstract) **vii**

1 Introduction **1**
- 1.1 Related work . . . 1
- 1.2 Contribution . . . 2

2 Foundations **3**
- 2.1 Alternative graph model . . . 3
- 2.2 Job-Shop scheduling . . . 3
  - 2.2.1 Job scheduling problem . . . 4
  - 2.2.2 Job-Shop scheduling problem . . . 4
  - 2.2.3 Blocking constrains . . . 6
  - 2.2.4 Finding the best selection . . . 6
- 2.3 Conflict resolution problem with fixed routes . . . 8
  - 2.3.1 Problem description . . . 8
  - 2.3.2 Modeling as a job-shop scheduling problem with blocking constraints . . . 8
  - 2.3.3 Static implications . . . 9
- 2.4 RAS Problem Solving Competition 2022 . . . 12
  - 2.4.1 Infrastructure model . . . 13
  - 2.4.2 Timetable model . . . 15
  - 2.4.3 Problem instances . . . 17
  - 2.4.4 Timetable amendments . . . 17
  - 2.4.5 Penalty computation . . . 19
  - 2.4.6 Hard constrains . . . 21

3 Approach **27**
- 3.1 Modeling the RAS problem using an alternative graph . . . 27
  - 3.1.1 Extending alternative pairs . . . 27
  - 3.1.2 Basic model . . . 28
  - 3.1.3 Stop/Pass choices . . . 30
  - 3.1.4 Variable runtimes . . . 31
  - 3.1.5 Variable link headways . . . 32
  - 3.1.6 Track headways . . . 36
  - 3.1.7 Limiting early departures . . . 39
  - 3.1.8 Respecting future incidents . . . 39
  - 3.1.9 Respecting the realized schedule . . . 40
  - 3.1.10 Limits of the model . . . 42
  - 3.1.11 Extracting the amended timetable from a selection . . . 42

3.2 Static implications . . . 43
3.2.1 In the same direction . . . 43
3.2.2 In different directions . . . 44
3.2.3 For duty consistency . . . 45
3.3 Adapting the branch-and-bound algorithm . . . 46
3.3.1 Computing lower bounds . . . 46
3.3.2 Choice selection . . . 48
3.3.3 Search strategy . . . 50
3.3.4 Finding an initial complete selection . . . 50
3.4 Updating state metadata . . . 51
3.4.1 Topological order & longest paths . . . 52
3.4.2 Skipped stops . . . 52
3.4.3 Choice selection . . . 54
3.4.4 Early pruning & memory reusing . . . 56
3.4.5 Saving memory . . . 59
3.5 Speed-up techniques . . . 60
3.5.1 Predetermining alternatives . . . 60
3.5.2 Limiting the extent . . . 62

4 Evaluation **65**
4.1 General information . . . 65
4.1.1 Setup . . . 65
4.1.2 Model . . . 65
4.1.3 Problem instances . . . 65
4.1.4 Parameters . . . 67
4.2 Graph size . . . 67
4.3 Comparison to FCFS . . . 69
4.4 Comparison between parameters . . . 70
4.5 Investigating abort rate . . . 72
4.6 Evaluation of RAS example problems . . . 74

5 Conclusion & Outlook **77**
5.1 Conclusion . . . 77
5.2 Outlook & future work . . . 77

Bibliography **87**

Chapter 1

Introduction

The transportation sector is a significant contributor to global greenhouse gas emissions. Trains, as a low-carbon form of mobility, provide a good alternative to cars and planes. However, the reliability and punctuality of trains continue to be a major concern for users, as they significantly affect the overall quality of service. The increasing passenger demand and limited tracks have further complicated the issue. Technical failures and other disturbances are often unavoidable. They lead to primary delays of single trains. These propagate through the network and cause conflicts with other trains, making the existing schedule infeasible or leading to secondary delays. Resolving these conflicts in real-time during operations is crucial to ensure the efficient and reliable functioning of the entire railway network. Historically, conflict resolution was handled by experienced human experts. However, in recent years more research focused on automating this process.

In this thesis, we extend a real-time train disposition system introduced by [DPP07] to handle more complex requirements and constraints from the real world. The additional requirements are taken from the 2022 RAS Problem Solving Competition [INF23]. We use the infrastructure and timetable data of London's Elizabeth Line to evaluate our solution, based on the target function supplied by the competition. We model the additional constraints using an alternative graph $G = (V, F, A)$.

1.1 Related work

This work is based on [DPP07]. There, D'Ariano et al. formulate a branch-and-bound algorithm to schedule trains in a railway network. They use an alternative graph to model the conflict resolution problem for trains as a job-shop scheduling problem. Their optimization goal is to minimize the maximum secondary delay across all scheduled trains. We further look at their approach in Chapter 2.

[Cac+14] gives an overview of different approaches for real-time railway rescheduling. There, Cacchiani et al. classify recent approaches into multiple categories. They differentiate between the plain timetable rescheduling and the additional rescheduling of rolling stock. They also group the approaches into whether they handle disturbances or disruptions. Disturbances describe the fact that some railway operations such as dwelling or driving take longer. On the contrary, disruptions are incidents that have a big impact on the entire network, such as the unavailability of certain infrastructure. Lastly, they differentiate between whether the railway system is considered on a macroscopic or microscopic level.

In [BSV17], Bettinelly et al. employ a time-space graph for modeling the train rescheduling problem. Initially, they represent the railway network as a network graph denoted as $G_N = (V, A)$. This representation can accommodate both microscopic and macroscopic views of the underlying infrastructure. To incorporate the temporal dimension, they expand the graph by duplicating the vertices for each time instant, following a predetermined time discretization. The edges connecting vertices at different time instants capture the required runtime between corresponding resources. Then, a schedule is represented by a collection of paths within the time-space graph G_{TS}. Finally, an iterated greedy algorithm is employed to determine the optimal schedule that satisfies all imposed constraints.

Shakibayifar et al. propose a multi-objective real-time decision support system for train rescheduling in [SSJ18]. They calculate an initial solution by dynamic priority dispatching. Therefore, they calculate a dynamic priority for every train involved whenever a conflict occurs. It is based on the train's class and its delay up to the conflict. The train with the highest priority has the right of way. They improve this initial solution using a variable neighborhood descent algorithm. A proposed multi-objective version of this algorithm is capable of finding the Pareto fronts for multiple objective functions. The two objective functions used are the sum of the destination delays of all trains and the sum of the deviations from the original schedule at relevant points in the network.

1.2 Contribution

Our contribution is the extension of existing models to incorporate more complex constraints and target functions. Existing models based on the alternative graph only consider the maximum delay across all trains. We use the sum of multiple delays instead. Furthermore, our model allows for more degrees of freedom when rescheduling trains. For instance, the decision of whether a train dwells at a station or skips it. It also involves a local rerouting directly in the branch-and-bound algorithm. We do this by selecting the track that trains use at a station dynamically. Additionally, we consider dynamic headway constraints that depend on those decisions. All in all, we apply the alternative graph model to a real-world problem with elaborate constraints.

Moreover, we introduce a concise mathematical formalization of the 2022 RAS problem. The original problem description is partially informal.

Chapter 2

Foundations

This chapter introduces the fundamentals necessary for our approach. We start by introducing alternative graphs in Section 2.1. Then, in Section 2.2, we show how alternative graphs can be used for solving the job-shop scheduling problem. After that, Section 2.3 introduces the conflict resolution problem with fixed routes for trains (CRFR). In the same section, we also show how the CRFR problem can be reduced to a job-shop scheduling problem. Finally, in Section 2.4 we introduce the 2022 RAS Problem Solving Competition. The competition defines the problem that is solved by our approach.

2.1 Alternative graph model

An alternative graph is a triple $G = (V, F, A)$. It contains a set of vertices V, a set of fixed edges $F \subseteq V^2$, and a set of pairs of alternative edges $A \subseteq V^2 \times V^2$. The latter are also called alternative pairs. The alternative pairs are of the form $p = (a_0, a_1)$ with $a_0 = (i, j)$ and $a_1 = (h, k)$. We say that a_0 and a_1 are the two alternatives of p. We also refer to a_0 as the alternative of a_1. Both the fixed and the alternative edges are weighted. A selection $\mathcal{S}$ is a set of alternative edges. It may not contain more than one alternative edge from every alternative pair in A. The graph induced by a selection $\mathcal{S}$ is $G(\mathcal{S}) = (V, F \cup \mathcal{S})$. A selection is *consistent* if $G(\mathcal{S})$ does not contain any positive length cycles. A selection is *complete* if it contains exactly one alternative from every pair in A. Otherwise, the selection is *partial*. For a selection $\mathcal{S}$, an *extension* of $\mathcal{S}$ is a complete consistent selection $\mathcal{S}'$ such that $\mathcal{S} \subseteq \mathcal{S}'$. An alternative edge (i, j) is selected if $(i, j) \in \mathcal{S}$. An alternative pair $p = ((i, j), (h, k))$ is selected if either $(i, j) \in \mathcal{S}$ or $(h, k) \in \mathcal{S}$. Otherwise, they are unselected. The concept of alternative graphs was introduced in [Pac02].

Alternative graphs are a generalization of disjunctive graphs. Disjunctive graphs were first introduced by Roy et al. in [RS64]. They only allow for alternatives pairs of the form $((i, j), (j, i))$. Also, the fixed edges must form a set of distinct paths from a dedicated start to a dedicated end vertex.

We use alternative graphs in the following sections to model different scheduling problems.

2.2 Job-Shop scheduling

In this section, we introduce different scheduling problems and model them using alternative graphs. First, Section 2.2.1 defines and solves the job scheduling problem using a simple graph. Then, Section 2.2.2 and 2.2.3 introduce different job-shop scheduling problems and use alternative graphs to model them. Last, Section 2.2.4 presents a branch-and-bound approach for solving the job-shop scheduling problems.

$$
\begin{aligned}
J &= \{j_1, j_2, j_3\} \\
P &= \{(j_1, j_2), (j_1, j_3)\} \\
f &= \{((j_1, j_2), 5), ((j_1, j_3), 7)\} \\
d &= \{(j_1, 5), (j_2, 8), (j_3, 7)\} \\
st &= \{(j_1, 2), (j_2, 4), (j_3, 4)\}
\end{aligned}
$$

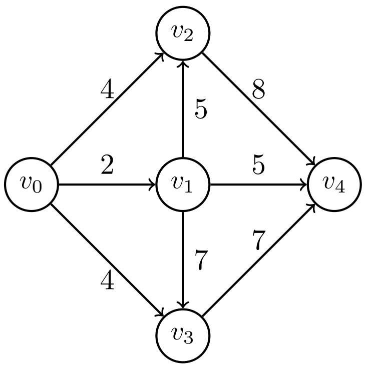


FIGURE 2.1: Example of transforming a job scheduling problem into a longest path problem. The original problem is shown on the left and the resulting graph on the right.

2.2.1 Job scheduling problem

The job scheduling problem is the problem of scheduling multiple jobs such that precedence constraints between them are fulfilled. The input is a set of jobs $J = \{j_1, \dots, j_n\}$, the processing time of each job $p : J \to \mathbb{N}$, and a set of precedence relations $P \subset J^2$. Each of those relations has a weight $w : P \to \mathbb{N}$. Here $(j_1, j_2) \in P$ means that j_1 must be scheduled $w((j_1, j_2))$ units before job j_2. For example $w((j_1, j_2)) = p(j_1)$ implies that j_2 can only start after j_1 was completed. Additionally, there is an earliest possible start time $st : J \to \mathbb{N}$ for each job. The output is a schedule that assigns each job a start time $S : J \to \mathbb{N}$. In the resulting schedule, no job may start before its earliest possible start time and all precedence relations must be fulfilled. The latest completion time of all jobs is the makespan of a schedule. Our optimization goal is to find a schedule with minimal makespan.

The problem can be transformed into a longest path problem. This means that it can be solved by finding the (length of) the longest path from a start vertex to all other vertices in a graph. For the transformation, we create a weighted directed graph $G = (V, E)$. First, we add two dummy jobs j_0 and j_{n+1} with a processing time of 0 to J. Those jobs model the start and respectively the end of the schedule. Also, for every original job $j \in J \setminus \{j_0, j_{n+1}\}$ we add the precedence relations (j_0, j) with weights $w((j_0, j)) = st(j)$ to P. Those relations model that each job is scheduled after its earliest possible start time. Furthermore, we add the relations (j, j_{n+1}) with weights $w((j, j_{n+1})) = p(j)$. They model that the dummy end job is scheduled after each job is finished. We set $V = \{v_i | j_i \in J\}$ and $E = \{(v_i, v_k) | (j_i, j_k) \in P\}$ with $w((v_i, v_k)) = w((j_i, j_k))$. Now, the longest distance to any vertex v_i from v_0 is the earliest possible start time of job j_i in a schedule respecting all precedence constraints. Respectively, the longest distance from v_0 to v_{n+1} is the resulting makespan of the schedule. Figure 2.1 shows an example of this mapping. In the following, we define $\text{lp}(v_i, v_k)$ to be the distance of the longest path from v_i to v_k. We also use $\text{lp}(v_k)$ if the origin of the path is clear from the context or the graph has a dedicated start vertex. $\text{lp}(v_i, v_k) = -\infty$ if there is no path from v_i to v_k.

2.2.2 Job-Shop scheduling problem

The following section is based on [MP02].

Just like the job scheduling problem, the job-shop scheduling problem is about finding a schedule with a minimal makespan. However, it has additional inputs and

$$\begin{aligned} M(o_{0,0}) &= M(o_{1,0}) \\ &= M(o_{2,0}) \\ M(o_{1,1}) &= M(o_{2,1}) \\ M(o_{0,2}) &= M(o_{1,2}) \end{aligned}$$

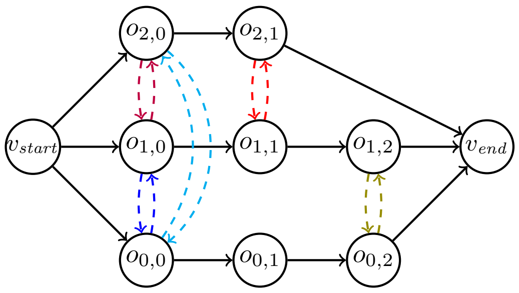


FIGURE 2.2: Example of the alternative graph for a job-shop scheduling problem

constraints. First, there is a set of machines M. The machines are also called shops, hence the name of the problem. Furthermore, each job consists of a set of operations. Each of those operations o can only be processed by a specific machine $m = M(o)$. Moreover, each machine can only process one operation at a time. The operations of a job are linearly ordered and have a processing time $p(o)$. We call the set of all operations $O = \{o_1, \dots, o_n\}$.

With the additional constraints, it is not possible to schedule all jobs in parallel. This is because of the limited resources of each machine. Also, the order of the operations at the machines is not clear in advance. Therefore, we can not model the problem by a simple directed graph. However, we can model it using an alternative graph.

A job-shop problem can be mapped to an alternative graph $G = (V, F, A)$ in the following way. The vertex set V is the set of operations. Additionally, it is enhanced by two dummy vertices modeling the start (v_{start}) and end of the schedule (v_{end}). We create a fixed edge from (v_{start}) to the first operation of each job j. Its weight is the job's earliest possible start time $\mathrm{st}(j)$. Additionally, we create fixed edges from the last operation o of each job to v_{end}. Each edge has the weight $p(o)$. We model the linear order of operations within a job as precedence relations. This means that we add fixed edges from every operation o to its successor. Their weight is the processing time of the prior operation $p(o)$. This ensures that the successor operation does not start before the prior has finished. Then, for every pair of operations (o, q) that need to be processed at the same machine, we add the alternative pair $((o, q), (q, o))$ to A. We assign the corresponding alternative edges the weights $w((o, q)) = p(o)$ and $w((q, o)) = p(q)$. This ensures that no machine is occupied by two operations at the same time. By using alternative edges, we do not force the order of operations in advance. Figure 2.2 shows an example of the resulting alternative graph. In the figure, dashed edges are alternative edges. If they have the same color, they are alternatives of the same pair.

With this modeling, we can map every consistent selection $\mathcal{S}$ to a valid schedule for the job-shop problem. The correspondence arises by solving the longest path problem in $G(\mathcal{S})$ (cf. Section 2.2.1). Also, the schedule that minimizes the makespan must correspond to a consistent selection. Otherwise, it would include redundant wait times that made it non-optimal. As a consequence, the job-shop problem can be solved by finding the selection that minimizes the makespan of the resulting schedule. In the following we define $\mathrm{lp}_s(v_0, v_1)$ to be the length of the longest path from v_0 to v_1 in $G(\mathcal{S})$. We again also write $\mathrm{lp}_s(v_1)$ if the origin of the path is clear from the context or if the graph has a dedicated start vertex.

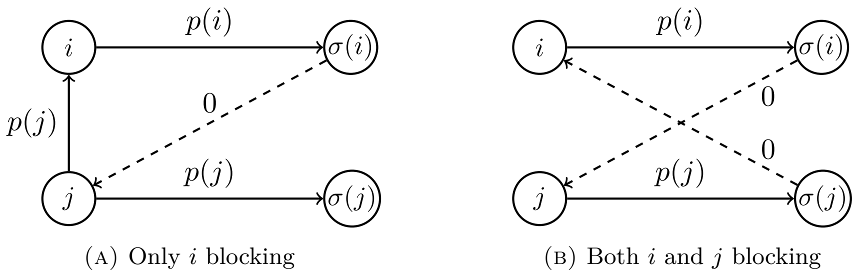


FIGURE 2.3: Blocking constraints in the alternative graph. Dashed edges are alternative edges.

2.2.3 Blocking constrains

The graph constructed in Section 2.2.2 is a disjunctive graph. Disjunctive graphs suffice to model the basic job-shop scheduling problem. However, they fail to capture some more complicated constraints from real-world applications. This section introduces one of those constraints and shows how to model it using an alternative graph. The section is again based on based on [MP02].

The current model assumes that when an operation is finished at a machine, the machine is immediately available for a new job. Yet, in a factory setting, there may be no possibility to store the product of a job between two operations. In this case, the job occupies the first machine until the machine of the successor operation becomes available. Such a constraint is called *blocking*. We also say that operations that occupy their machine until the next becomes available are *blocking* operations. Operations that are not blocking but leave the machine after their processing time are called *ideal*.

We can model blocking operations in the following way. Let $i, j \in O$ be two operations of different jobs at the same machine. Let $\sigma(o)$ be the successor of an operation in the same job. If i is a blocking operation and starts before j, j can only start after $\sigma(i)$ started. This is modeled by the edge $(\sigma(i), j)$ with weight 0. If j starts before i, there are two possibilities. Either j is also blocking. Then, this can be modeled symmetrically by the edge $(\sigma(j), i)$. If j is ideal, the constraint is modeled as in the disjunctive graph. Namely, with the edge (j, i) with weight $p(j)$. To not predetermine the order of the operations, those edges must be the alternatives of the same alternative pair. Consequently, we either add the alternative pair $((\sigma(i), j), (\sigma(j), i))$ or $((\sigma(i), j), (j, i))$ to A. Figure 2.3 shows both cases. In any case, the resulting graph is no longer a disjunctive graph.

Also, to make the model well-defined, we add a new ideal operation to every job. This new operation has no processing time and is preceded by all other operations in the job. Also, it must be processed by a machine unique to the operation. With these new operations, $\sigma(o)$ is defined for all blocking operations. However, no new constraints or processing times are introduced.

2.2.4 Finding the best selection

Given an alternative graph $G = (V, F, A)$, finding the selection that minimizes the makespan in $G(\mathcal{S})$ is a non-trivial task. A simple brute-force approach needs an exponential amount of time as there are $2^{|A|}$ possible selections. To find the optimal selection faster, we need to explore the search space more efficiently. This section describes how a branch-and-bound algorithm can be used to achieve this.

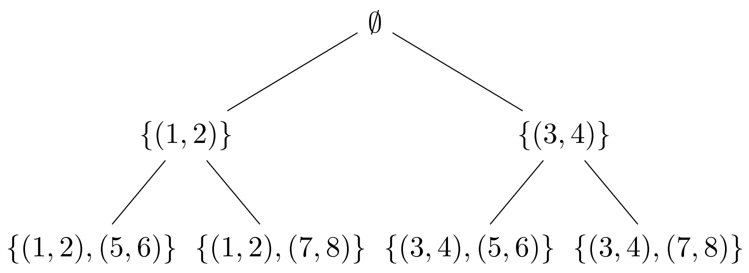


FIGURE 2.4: Example of a simple branch and bound search tree with $A = \{((1,2),(3,4)),((5,6),(7,8))\}$

Branch-and-bound algorithms were formalized by Lawler et al. in [LW66]. They explore the search space by searching through a rooted tree. Here, every leaf node represents a solution to the problem. An inner node represents a subset of all solutions. More specifically, it represents the set of all solutions that are represented by the leaf nodes of the subtree rooted at the inner node. The structure of the tree and how it is explored depends on the specific problem. The efficiency gains compared to a brute-force approach are achieved by computing lower bounds at intermediate nodes. Those are used to prune entire subtrees. For every node, a lower bound is a value smaller than the value of the objective function of any of its solutions. Meaning, that every solution that is contained in the subtree rooted at the node has a worse objective function value. Additionally, we always store the best solution we found so far and its objective function value. Therefore, if the best solution we found is better than the computed lower bound, we may prune the whole subtree. This means, that we no longer spend time exploring it. If we cannot prune a subtree, we branch on it by considering the child nodes of the previous node individually. If we reach a leaf that represents a better solution than the current best, we store it as the new current best solution. Otherwise, we discard it. The algorithm starts the exploration at the root of the tree.

In our case, the leaf nodes are complete consistent selections. The inner nodes are partial selections. They represent all their possible extensions. The root node is the empty selection $\emptyset$. A parent node is always a subset of each of its children. Every edge in the tree represents adding another alternative edge to the selection. Let $\mathcal{S}$ be a partial selection. We can compute a lower bound for the makespan of all its extensions $\mathcal{S}'$ in a straightforward way. Notice that the makespan of the schedule corresponding to $\mathcal{S}'$ is just the longest path between the start and end vertex in the graph $G(\mathcal{S}')$. More formally, $\mathrm{lp}_{\mathcal{S}'}(v_{start}, v_{end})$. Also, notice that the length of the longest path between two vertices can only increase by adding edges to a graph. However, $G(\mathcal{S}')$ only differs from $G(\mathcal{S})$ by adding the edges $e \in \mathcal{S}' \setminus \mathcal{S}$. Therefore, $\mathrm{lp}_{\mathcal{S}}(v_{start}, v_{end}) \leq \mathrm{lp}_{\mathcal{S}'}(v_{start}, v_{end})$ is a lower bound for the makespan of $\mathcal{S}'$.

When branching, it is not clear which alternative pair should be considered next. However, the structure of the search space is highly dependent on this decision. For efficient computation, a well-structured search space is important. Whether it is well structured, must be checked by experiments with the concrete problem at hand. The approach chosen by [DPP07] is to always branch on the alternative pair that has the alternative that increases the makespan the most. The intention behind it is to branch early on alternative pairs that have a high impact on the makespan. Figure 2.4 shows an example of a simple search tree.

2.3 Conflict resolution problem with fixed routes

The conflict resolution problem with fixed routes (CRFR) is the problem of finding a conflict-free train schedule given a disturbed timetable. We further define, model, and solve it in the following sections. First, Section 2.3.1 introduces the CRFR problem. Then, in Section 2.3.2, we model it using an alternative graph by viewing it as a job-shop scheduling problem. Last, Section 2.3.3 introduces so-called *static implications* and explains their computation. They are a method for speeding up the branch-and-bound algorithm for finding the optimal selection. The whole section is based on [DPP07].

2.3.1 Problem description

On a microscopic level, railway networks usually consist of tracks and signals. Signals are used to convey information to the train to avoid collisions. Block signals signal whether a train may enter the line ahead. Sections between two block signals are called block sections. There may only be one train in a block section at a time. Trains need a certain amount of time for crossing each block section, called *running time*. After a train enters the next block section, the previous section stays blocked for some time for safety reasons. In [DPP07], D'Ariano et. al refer to this time as *setup time*. A conflict appears when more than one train requires the same block section at the same time. The conflict resolution problem is the problem of avoiding such conflicts given the real-time state of the network. The optimization goal is to minimize the maximum delay of trains at dedicated points in the network.

Formally, we have a set of trains T and a set of block sections B. Each train $t \in T$ must pass a sequence of block sections $(s_0, \ldots, s_{n-1}) \in B^n$. Additionally, there is a planned arrival and departure time for each train at those block sections $\text{arrival} : T \times B \to \mathbb{N}$ and $\text{departure} : T \times B \to \mathbb{N}$. The real-time state of the network is represented by the earliest possible time a train may enter the first block section $st : T \to \mathbb{N}$. This time may be different than in a pre-planned timetable. Then, the problem is non-trivial, as the pre-planned timetable is no longer a viable solution. As a consequence, we have to adapt the departure and arrival times such that there is no conflict between two or more trains. Furthermore, the new arrival and departure times must be consistent with running and setup times. We want to do this such that the maximum delay, compared to the original arrival and departure times, is minimized. For the maximum delay, we consider the delays the trains have when they leave the considered railway network. Additionally, we also consider the arrival delay at a subset of the block sections $B_d \subseteq B$. Depending on the scenario, those might represent stations.

The problem is called conflict resolution problem with *fixed routes* because we do not modify the sequence of block sections each train has to pass. In some cases, there might be alternative routes through the network that are also viable for real-time disposition. However, in this section, we restrict our view to the fixed routes version of the conflict resolution problem.

2.3.2 Modeling as a job-shop scheduling problem with blocking constraints

We can transform the conflict resolution problem with fixed routes into a job-shop scheduling problem with blocking constraints. The latter was introduced in Section 2.2.3. For the transformation, we create a job for every train $t \in T$. Then, every block section the train has to pass is an operation of this job. Apart from that, the set of machines M is defined to be the set of block sections B. This correspondence matches, as every block section may only host a single train at a time.

The machines needed for processing the operations are assigned as follows. Let $(s_0, \dots, s_{n-1}) \in B^n$ be the block sections of a train $t \in T$. Also, let $(o_0, \dots, o_{n-1})$ be the operations of the corresponding job. Then, $M(o_i) = s_i$. This means that the machine of the operation, which corresponds to a train passing a block section, is the block section itself. The processing time of an operation $p(o)$ is defined by the corresponding running time of the train through the block section. Furthermore, every operation is blocking. This is because the block sections can only be used again after a prior train has successfully entered the next block section.

The setup times are modeled by the weights of alternative edges. For two blocking operations i, j with $M(i) = M(j) = s$ we have the alternative edge $((\sigma(i), j), (\sigma(j), i))$. In the current model, those edges have a weight of 0. This models that the second operation can start as soon as the first operation leaves the machine. However, with the additional constraint of setup times, this is not correct. Therefore, let $su_{i,j}$ be the setup time at the block section s between the train of operation i and the train of operation j. Then, $w((\sigma(i), j)) = su_{i,j}$ and, symmetrically, $w((\sigma(j), i)) = su_{j,i}$.

We want to minimize the maximum delay at different points of the network. However, in our current model, the makespan corresponds to the time when the last train has passed all its block sections. To change this, we adapt the weight of the fixed edge from the last operation $o_{i,last}$ of each train t_i to the end vertex v_{end}. For example, let δ_i be the originally scheduled time train $t_i \in T$ completes its last operation. If we set the weight $w((o_{i,last}, v_{end}))$ to $-\delta_i$, $\mathrm{lp}_S(v_{start}, o_{i,last}) + w((o_{i,last}, v_{end}))$ becomes the delay that t_i has when leaving the network. If we do this for all trains, the makespan $\mathrm{lp}_S(v_{start}, v_{end})$ becomes the maximum of those delays. Additionally, let o be an operations of train t with $M(o) = b \in B_d$. Then, we add a fixed edge (o, v_{end}) to the graph. Its weight $w((o, v_{end}))$ is $-\mathrm{arrival}(t, b)$. As a consequence, the maximum delay now also considers the arrival delay of all trains at the block sections $b \in B_d$.

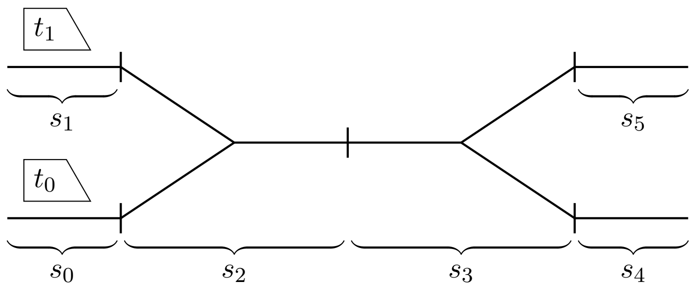
Sometimes trains may not leave a block section before a certain time. We can also model this using the alternative graph. Our goal is to enforce the minimum completion time of an operation o. We can achieve this by adding fixed edges from the start vertex to $\sigma(o)$. The weight of those edges is the specified minimum completion time mc_o. Then, $\mathrm{lp}_S(\sigma(o)) \geq mc_o$. This means that $\sigma(o)$ cannot start before mc_o. As o is blocking, this also implies that o does not complete before mc_o.

Figure 2.5 shows an example of mapping a conflict resolution problem to an alternative graph. Here, we have two trains $T = \{t_0, t_2\}$ and six block sections $B = M = \{s_0, \dots, s_5\}$. t_0 needs to pass the sequence (s_0, s_2, s_3, s_4) while t_1 needs to pass (s_1, s_2, s_3, s_5). Consequently, there might be conflicts for block sections s_2 and s_3. Because of this, the graph contains alternative edges between the corresponding operations. The edges colored in the same color are alternatives of the same alternative pair. There are no alternative edges for block sections that are not used by both trains.

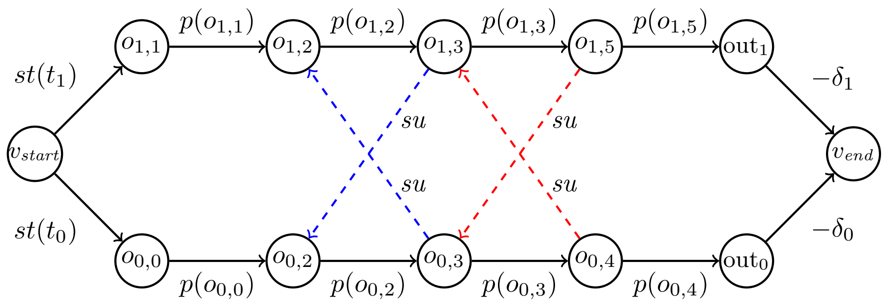
With this modeling, we can also solve the CRFR problem using the branch-and-bound approach introduced in Section 2.2.4.

2.3.3 Static implications

The branch-and-bound approach from Section 2.2.4 is capable of finding the best selection of general alternative graphs. However, we can speed it up by utilizing additional knowledge about our specific problem instance. To accomplish this, we use



(A) Railway network with 6 block sections and two trains



(B) Corresponding alternative graph

FIGURE 2.5: Modeling the Conflict Resolution Problem as an alternative graph with $B_d = \emptyset$

the topology of the underlying railway network to compute so-called *static implications*.

Let $(a_0, a_1) \in A$. If there is no extension of $\mathcal{S} \cup \{a_1\}$ that improves the current best solution, we say that a_0 is *implied* by a selection $\mathcal{S}$. We write $\mathcal{S} \rightsquigarrow a_0$. Then, in the branch-and-bound algorithm, we can immediately select a_0 without considering the other subtree. However, proving this for a given selection $\mathcal{S}$ can be time-consuming when dealing with a large number of jobs and machines. Implications that can only be computed for a given selection are called *dynamic*. Static implications can be computed up-front. They only depend on a single alternative. We say that an alternative a_i implies a_j if $\{a_i\} \rightsquigarrow a_j$. We also write $a_i \rightsquigarrow a_j$. Then, the implication is valid for all supersets of $\{a_i\}$. This is as for $S' \supseteq S$,

$$S \rightsquigarrow (i,j) \implies S' \rightsquigarrow (i,j)$$

In the following, we explain two static implication rules introduced in [DPP07]. They rely on the following theorem.

Theorem 2.3.1. *Given a selection $\mathcal{S}$ and two unselected pairs $((a,b),(c,d)) \in A$ and $((i,j),(h,k)) \in A$. Then*

$$\begin{aligned} w((a,b)) + \mathrm{lp}_S(b,i) + w((i,j)) + \mathrm{lp}_S(j,a) \geq 0 \implies & (S \cup \{(a,b)\} \rightsquigarrow (h,k)) \\ & \wedge (S \cup \{(i,j)\} \rightsquigarrow (c,d)) \end{aligned}$$

Proof. As $S \cup \{(a,b),(i,j)\}$ contains a positive length cycle, any extension cannot be consistent. □

For finding static implication rules, we use the theorem with $S = \emptyset$. For our specific alternative graph, $w((a,b)) + \mathrm{lp}_\emptyset(b,i) + w((i,j)) + \mathrm{lp}_\emptyset(j,a) \geq 0$ holds, if the following conditions are true.

1. b and i are operations of the same train $t_1 \in T$ and b must be completed by t_1 before i
2. j and a are operations of the same train $t_2 \in T, t_2 \neq t_1$ and j must be completed by t_2 before a
3. t_1 and t_2 pass through the two block sections $s_1, s_2 \in B$ for which the alternative pairs $((a,b),(c,d))$ and $((i,j),(h,k))$ were added to the graph.

There are only two scenarios in which those conditions hold.

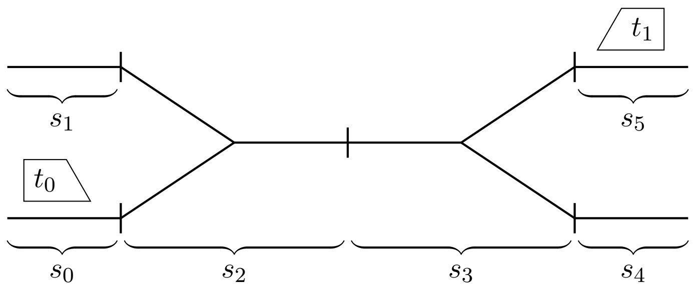
First, s_1, s_2 are adjacent and traversed by t_1, t_2 in the same order (cf. Figure 2.5a). Then, $a = j$ and $c = k$. From this follows $(a,b) \leftrightsquigarrow (h,k)$ and $(c,d) \leftrightsquigarrow (i,j)$. Here $(a,b) \leftrightsquigarrow (h,k) \Leftrightarrow (a,b) \rightsquigarrow (h,k) \wedge (h,k) \rightsquigarrow (a,b)$. For instance, for the alternative graph in Figure 2.5b the rule can be applied for $((a,b),(c,d)) = ((o_{1,3}, o_{0,2}), (o_{0,3}, o_{1,2}))$ and $((i,j),(h,k)) = ((o_{0,4}, o_{1,3}), (o_{1,5}, o_{0,3}))$. In the figure, we can see, that the addition of both $(o_{1,3}, o_{0,2})$ and $(o_{0,4}, o_{1,3})$ to any selection causes a cycle. Table 2.1 gives an overview of the implications and equalities described above. Descriptively, the rule models that trains may not overtake each other.

Equalities	Implications
$a = j = o_{1,3}$	$(o_{1,3}, o_{0,2}) \leftrightsquigarrow (o_{1,5}, o_{0,3})$
$b = o_{0,2}$	$(o_{0,3}, o_{1,2}) \leftrightsquigarrow (o_{0,4}, o_{1,3})$
$c = k = o_{0,3}$	
$d = o_{1,2}$	
$i = o_{0,4}$	
$h = o_{1,5}$	

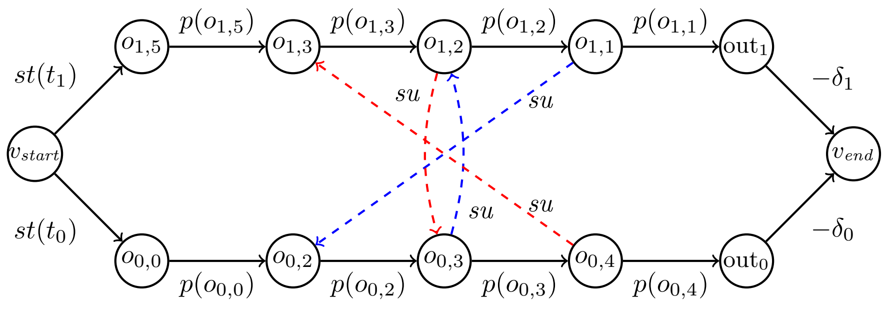
Table 2.1: Overview of equalities and implications in first case

The second scenario is that s_1, s_2 are passed by t_1, t_2 in opposite order (cf. Figure 2.6a). Then, $(a,b) \rightsquigarrow (h,k)$ and $(i,j) \rightsquigarrow (c,d)$. However, $(h,k) \not\rightsquigarrow (a,b)$ and $(c,d) \not\rightsquigarrow (i,j)$. In the example from Figure 2.6b we can apply the rule for $((a,b),(c,d)) = ((o_{1,1}, o_{0,2}), (o_{0,3}, o_{1,2}))$ and $((i,j),(h,k)) = ((o_{0,4}, o_{1,3}), (o_{1,2}, o_{0,3}))$. We can see in the figure, that the addition of $(o_{1,1}, o_{0,2})$ and $(o_{0,4}, o_{1,3})$ to a selection causes a cycle. However, a consistent selection may contain both $(o_{1,2}, o_{0,3})$ and $(o_{0,3}, o_{1,2})$. Table 2.2 gives an overview of the implications and equalities described above. Descriptively, the rule models that a train t_1 can not reach a block section b before another train t_2 which is going in the reverse direction if the t_2 reaches t_1's way to b first.

Using the rules introduced above, we assign each alternative edge (i,j) a set of alternative edges that are statically implied by the edge. We call this set $\mathrm{Stat}((i,j))$. Now, whenever we add an edge (i,j) to the current selection in the branch-and-bound algorithm, we also add all edges in $\mathrm{Stat}((i,j))$ to the selection. We do this recursively for the edges added this way. We call the recursively closed set of edges added this way $\mathrm{Stat}^*((i,j))$.



(A) Railway network with 6 block sections and two trains



(B) Corresponding alternative graph

FIGURE 2.6: Modeling the Conflict Resolution Problem as an alternative graph. Two trains use the same block sections in different directions

Equalities	Implications
$a = o_{1,1}$	$(o_{1,1}, o_{0,2}) \rightsquigarrow (o_{1,2}, o_{0,3})$
$b = o_{0,2}$	$(o_{0,4}, o_{1,3}) \rightsquigarrow (o_{0,3}, o_{1,2})$
$c = o_{0,3}$	
$d = o_{1,2}$	
$i = o_{0,4}$	
$j = o_{1,3}$	
$h = o_{1,2}$	
$k = o_{0,3}$	

TABLE 2.2: Overview of equalities and implications in second case

2.4 RAS Problem Solving Competition 2022

The RAS Problem Solving Competition is an annual competition held by the Railways Application Section (RAS) of the Institute for Operations Research and the Management Sciences (INFORMS)[1]. Each year, the competition presents a problem with practical relevance for either passenger or freight rail. [INF23] gives an overview of all prior problems, as well as the original problem description for the 2022 competition. The latter defines the problem solved by our approach. Its focus is to design a real-time traffic management system for London's Elizabeth Line.

The Elizabeth Line is a railway line that stretches over 100 km, connecting the east and west of London. Its structure resembles an 'X', with a unified central tunnel that splits into two branches on the east and west sides. The western terminals are Reading and Heathrow Airport, while the eastern line extends to Shenfield and Abbey Wood.

[1]https://connect.informs.org/railway-applications/home

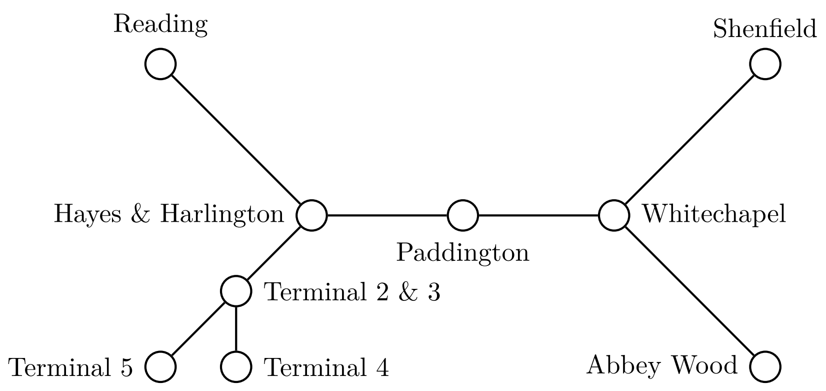


FIGURE 2.7: General Structure of the London Elizabeth Line

Figure 2.7 gives an overview of the general structure of the network. On each day, the line carries more than 1000 trains. During peak hours, the stations in the central section between Paddington and Whitechapel are served up to every 150 seconds in each direction. With such a high service frequency, even minor disturbances may cause significant secondary delays and timetable disruptions.

The competition assigns such timetable disruptions a monetary cost. The goal of the underlying optimization problem is to minimize the cost of the amended timetable. Apart from the infrastructure and original timetable, the input to the problem is a specific point in time, the realized timetable up to that point, and a set of expected future incidents that occur after that time.

The subsequent sections provide a more detailed description of the problem. Section 2.4.1 focuses on the underlying infrastructure model. Then, Section 2.4.2 introduces the timetable and rolling stock model. Section 2.4.3 provides more information on the specific problem instances and potential future incidents. Section 2.4.4 outlines the possible timetable amendments and defines the amended timetable. Then, Section 2.4.5 explains how the monetary penalty is calculated based on an amended timetable. Finally, Section 2.4.6 lists the hard constraints that apply to the amended timetable.

2.4.1 Infrastructure model

The infrastructure is given by a macroscopic model based on a simple directed graph $\mathcal{G} = (N, L)$. Here N is a set of nodes. Each node belongs to one of the three categories station, junction, and control point. Stations are the nodes where passengers can enter and exit trains. We model this by the function $\text{ncat} : N \to \mathit{NodeCategory}$ and the set $\mathit{NodeCategory} = \{\mathit{Station}, \mathit{Junction}, \mathit{ControlPoint}\}$. Additionally, each node has two sets of tracks. One for the eastbound direction and one for the westbound direction. Those are the tracks that can be used at the node by a train going in the respective direction. The two sets may overlap. We model a track t as a natural number. We define the function $\text{tracks} : N \times \mathit{Direction} \to 2^{\mathbb{N}}$ for modeling the tracks of a node. We also define the set $\mathit{Direction} = \{\mathit{Eastbound}, \mathit{Westbound}\}$.

Between the nodes are links $l \in L \subseteq N^2$. Each link represents a single track between two nodes. It has a direction assigned to it. The possible directions are *Eastbound*, *Westbound*, and the special direction *Both*. Eastbound or Westbound links can only be used by trains traveling in the respective direction. Their track is separated from the track of the reverse link. For example, one train could use the link (A, B) at the same time as another train uses the link (B, A). Links with the direction

Both have different semantics. They only occur to and from nodes with degree one. If the link (A, B) has direction *Both* then also (B, A) has direction *Both*. Trains can use those links independently of the direction they are traveling in. However, their track is not separated from the track of the reverse link. This means if (A, B) and (B, A) have direction *Both*, then they can not be used concurrently. They model the same physical track. Nonetheless, most of the network is double-track. Also, there is never more than one link with the same direction between two nodes.

We model a link's direction with the function $\mathrm{ldir} : L \to \mathit{Direction}^*$. Here $\mathit{Direction}^*$ is the set $\mathit{Direction} \cup \{\mathit{Both}\}$.

Each link is also assigned a minimum runtime. It is the minimum possible time a train needs to drive from the link's start node to its end node. It is measured in seconds and depends on the activity at the start and end nodes. The possible activities for trains at nodes are *Stop* and *Pass*. *Stop* means that the train halts at a node. Then, passengers can enter and exit the train. *Pass* means that the train drives through the node without halting. For modeling, we define the set $\mathit{Activities} = \{\mathit{Stop}, \mathit{Pass}\}$ and the function $\mathrm{mrt} : L \times \mathit{Activities}^2 \to \mathbb{N}$. $\mathrm{mrt}((A, B), \mathit{Stop}, \mathit{Pass}) = 43$ means that the minimum runtime for a train traveling from A to B is 43 seconds if that train stops at node A and passes through node B.

Furthermore, there is a minimum headway for each link. It is defined as the minimum time necessary between two trains entering a link. It is again measured in seconds and depends on the activities of the involved trains. Those are the activities of the first train at the link's start and end node and the activities of the following train at those nodes. Consequently, we define the function $\mathit{LHW} : L \times \mathit{Activities}^4 \to \mathbb{N}$. $\mathit{LHW}((A, B), \mathit{Stop}, \mathit{Stop}, \mathit{Pass}, \mathit{Stop}) = 50$ means that the minimum time between two trains leaving node A towards B is 50 seconds if the first train stops at both A and B and the following train passes through A and stops at B.

Apart from the link headways, there is also a headway restriction at nodes. If two trains use the same track at a node, there must be a 30-second minimum headway between the first train leaving the node and the following train entering it. Also, links with direction *Both* impose another kind of headway. If (A, B) and (B, A) have direction *Both*, then between a train leaving link (A, B) and a train entering (B, A), there must be a 30-second minimum headway. A more detailed specification of the additional headway constraints is part of Section 2.4.6.

Apart from the specifications so far, each node has *change-end times* assigned to it. Those are the minimum times a train needs to wait between serving two different courses[2]. They depend on the direction of the prior and next course. Therefore, we model this by the function $\mathrm{cet} : N \times \mathit{Direction}^2 \to \mathbb{N}$.

Furthermore, nodes with the type *Station* have so-called base station values assigned to them. The base station value is an arbitrary number that is needed for the penalty computation. Conceptually, it models the importance of a station, compared to the other stations. It depends on a direction and the time of day. For modeling, we define the function $\mathrm{bsv} : N \times \mathit{Direction} \times \mathbb{N} \to \mathbb{N}$. If $\mathrm{ncat}(n) \neq \mathit{Station}$ then $\mathrm{bsv}(n, d, t) := 0$.

Some nodes allow for short turning. This allows trains to change their direction at the node. This may only be possible from a subset of the tracks or even from the tracks in the opposite direction. It also depends on the direction of the train entering the node. Therefore, we define $\mathrm{st} : N \times \mathit{Direction} \to 2^{\mathbb{N}}$. Here $\mathrm{st}(n, \mathit{Eastbound})$ is the set of tracks that can be used for short-turning trains driving in *Eastbound* direction.

[2]Courses are defined in Section 2.4.2

Also, some of the nodes are depots. Depots are places where trains can be stored without blocking tracks in the remaining network. This is modeled by the set $DEP \subseteq N$. Lastly, the competition defines reference stations for measuring the service frequency. They are also needed for the penalty computation. As the frequency is always measured in a direction, we model them as the set $FMS \subseteq N \times Direction$.

Concluding, the complete infrastructure model is given by the tuple

$$\mathcal{I} = (\mathcal{G}, \text{ncat}, \text{tracks}, \text{st}, \text{ldir}, \text{mrt}, \text{lhw}, \text{bsv}, DEP, FMS)$$

2.4.2 Timetable model

The timetable is given by a set of courses C. A course models a scheduled train in the timetable. On each service day, a course is served by single rolling stock. Courses have a direction (either *Eastbound* or *Westbound*) and one of two categories. The first category is *Passenger*. It describes that the course can be used by passengers. The second category is *Empty*. Courses in this category are used for shuffling trains through the network. They can not be entered by passengers. We model this using the set $CourseCategory = \{Passenger, Empty\}$ and the function $\text{ccat} : C \to CourseCategory$. We also define the function $\text{cdir} : C \to Direction$.

Each course c has a schedule S_c. It describes the path a course takes through the network. The schedule is a non-empty sequence of schedule items $S_c = (si_0, \ldots, si_{m-1})$. Each *schedule item* is a quintuple $si = (n, arr, dep, act, t)$. It models the planned behavior of a train at a node. $n \in N$ is the node of the schedule item. $arr, dep \in \mathbb{N} \cup \{-1\}$ model the planned arrival and departure time at the node. They are given in seconds since midnight. In some cases, they might be greater than 86400, which is the number of seconds in a day. This is if the corresponding course still belongs to the last service day but continues after midnight. Moreover, there is no arrival time for the first and no departure time for the last schedule item of the sequence. This is because the arrival at the first and the departure from the last node are not part of the course. To model this, we use the special value -1. $act \in Activity$ models the planned activity of the course at the node. If the activity is *Pass*, the departure time and arrival time are the same. Otherwise, if the activity is *Stop* their difference is the planned dwell time at the node. $t \in \text{tracks}(n, \text{cdir}(c))$ models the track that shall be used at n. For convenience, we define some functions. Let $si_i = (n, arr, dep, act, t)$ be the i-th element of S_c, then

$$\begin{aligned}
\text{node}(si_i) &= n \\
\text{arrival}(si_i) &= arr \\
\text{departure}(si_i) &= dep \\
\text{activity}(si_i) &= act \\
\text{track}(si) &= t \\
\text{course}(si_i) &= c \\
\text{index}(si_i) &= i \\
\text{succ}(si_i) &= \begin{cases} si_{i+1} & \text{if } i < |S_c| - 1 \\ \emptyset & \text{otherwise} \end{cases}
\end{aligned}$$

Apart from courses, the timetable model also consists of a set of rolling stock duties R. Those are the duties a single train set, also called rolling stock, has to fulfill during the service day. Each rolling stock duty $d \in R$ is a non-empty sequence of rolling stock duty elements $d = (de_0, \dots, de_{m-1})$. Each rolling stock duty element is a sextuple

$$\begin{aligned}
&de = (\mathit{type}, \mathit{start\_node}, \mathit{end\_node}, \mathit{start\_time}, \mathit{end\_time}, \mathit{course})\\
&de \in (\mathit{RSTypes}, N, N, \mathbb{N}, \mathbb{N}, C \cup \{None\})
\end{aligned}$$

start_node and *end_node* are the nodes where the duty element starts and ends respectively. *start_time* and *end_time* model the planned start and end of a duty in seconds since midnight. *type* is the type of the duty element. There are four different types $\mathit{RSTypes} := \{Train, ChangeEnd, Spare, Reserve\}$. *Train* models that the rolling stock is operating a course. Then, $\mathit{course} \in C$ is the operated course. *start_node* is the node of the first schedule item of S_{course}. Symmetrically, *end_node* is the node of the last schedule item of S_{course}. Also, *start_time* is the departure time of the first schedule item of S_{course} and *end_time* is the arrival of the last schedule item of S_{course}. The second duty element type is *ChangeEnd*. It models the necessary processes between the operation of two courses. This is, for example, the process of a train driver going to the other cabin if the train changes directions between courses. The necessary time for those is the difference $\mathit{start\_time} - \mathit{end\_time}$. The third type is *Spare*. Rolling stock duty elements with this type model buffer time in the timetable. In the amended timetable, their duration can be reduced to 0 if necessary. A *Spare* duty element may follow a *ChangeEnd* duty element to make the schedule more resistant to potential delays. The last type is *Reserve*. It signals that the rolling stock is sitting idle at a depot. Therefore, it can be used to manage disruptions in the amended timetable. If the type of a rolling stock duty element is *Reserve*, it is the only element of the rolling stock duty. Then, we call the corresponding duty a reserve duty. All duty elements, except those with type *Train*, start at the same node where they end. Between two duty elements with type *Train*, there is always a *ChangeEnd* duty element and potentially a *Spare* element.

Again, we define some functions for convenience. Let

$$de_i = (\mathit{type}, \mathit{start\_node}, \mathit{end\_node}, \mathit{start\_time}, \mathit{end\_time}, \mathit{course})$$

be the i-th duty element of the rolling stock duty d, then

$$\begin{aligned}
\mathrm{type}(de_i) &= \mathit{type}\\
\mathrm{startNode}(de_i) &= \mathit{start\_node}\\
\mathrm{endNode}(de_i) &= \mathit{end\_node}\\
\mathrm{startTime}(de_i) &= \mathit{start\_time}\\
\mathrm{endTime}(de_i) &= \mathit{end\_time}\\
\mathrm{course}(de_i) &= \mathit{course}\\
\mathrm{index}(de_i) &= i
\end{aligned}$$

Furthermore, we define $\text{duty} : C \to R$. For a course $c \in C$ and a rolling stock duty $d \in R$ we have

$$\text{duty}(c) = d \iff \exists de \in d : \text{course}(de) = c$$

Concluding, the complete timetable model is given by

$$\mathcal{T} = (C, \text{cdir}, \text{ccat}, (S_c)_{c \in C}, R)$$

2.4.3 Problem instances

Based on the underlying infrastructure and timetable model, the 2022 RAS Problem Solving Competition formulates a problem instance. Each problem instance consists of a snapshot time t, a realized schedule until t and sets of known future incidents. It can be modeled by the sextuple

$$\mathcal{P} = (t, (RS_c)_{c \in C}, ERT, LD, ESD, ECD)$$

RS_c is the realized schedule for a course c. It is a (potentially empty) sequence of schedule items. This sequence contains schedule items that already happened before the snapshot time. We call them *realized* schedule items. They are related to the planned schedule items in S_c. Let $S_c = (si_0, \ldots, si_{k-1})$ and $RS_c = (rsi_0, \ldots, rsi_{\ell-1})$. Then, $\ell \leq k$ and $\text{node}(si_i) = \text{node}(rsi_i)$ for $i < \ell$. However, the arrival and departure times and track may differ between si_i and rsi_i. This represents delays and disruptions that already happened up until the snapshot time. The last realized schedule item of each course may not have a departure time. This means that the arrival happened before the snapshot time, but the departure did not. We again model this by setting $\text{departure}(rsi_{\ell-1}) = -1$. We call planned schedule items that do not have a correspondence in the realized schedule *non-realized*.

The known future incidents have different types. They are modeled by the four sets ERT, LD, ESD and ECD. First, $ert \in ERT \subseteq \mathbb{N}^2 \times L \times \mathbb{N}$ models an extended runtime on a link. This incident imposes that all trains using the specified link travel longer. For example, it might be caused by a temporary construction site at the link. Formally, $ert = (s, e, l, rt)$ where l is the link, s is the start time of the incident, e the end time and rt the runtime. If a train uses the link l in the interval (s, t) then its minimal runtime is rt. This is independent of the activities at the link's start and end nodes. Second, $ld \in LD \subseteq C \times \mathbb{N}$ models a late departure of an individual course. Formally, $ld = (c, d)$ where $c \in C$ is a course and $d \in \mathbb{N}$ a delay in seconds. It specifies that c departs d seconds later than planned in the original schedule. For instance, this might be because the assigned train driver is late. Third, $esd \in ESD \subseteq \mathbb{N} \times N \times \mathbb{N}$ models an extended dwell time at a station. This incident causes all trains, that stop at a specific node, to dwell longer. Formally, we have $esd = (s, e, n, dt)$. All trains stopping at n in the interval (s, e) dwell at least dt seconds. Trains that pass at n are not affected. Lastly, $ecd \in ECD \subseteq C \times N \times \mathbb{N}$ models an extended dwell time for a specific course at a node. Formally, $ecd = (c, n, dt)$. This incident causes the course c to dwell dt seconds at the node n instead of its normal dwell time.

All incidents happen after the snapshot time t.

2.4.4 Timetable amendments

Given the complete problem description $(\mathcal{I}, \mathcal{T}, \mathcal{P})$, our goal is to find an amended timetable that minimizes the objective function. An amended timetable is the quintuple $\mathcal{T}' = (C', \text{cdir}', \text{ccat}', (S'_c)_{c \in C'}, R')$. $C' \supseteq C$ is the set of amended courses. It may contain more courses than the original timetable. ccat′ and cdir′ are the extensions of ccat and cdir to C'. For $c \in C$, $\text{ccat}'(c) = \text{ccat}(c)$ and $\text{cdir}'(c) = \text{cdir}(c)$. S'_c is the amended schedule for a single course. It is again a sequence of schedule items. For $c \in C$, it can only differ from S_c by applying one or more of the following amendments:

Re-Timing Re-Timing allows for changing the planned arrival and departure times of a course at a node. Formally, if $si = (n, arr, dep, act, t)$ then $si' = (n, arr', dep', act, t)$.

Re-Platforming Re-Platforming allows for changing the scheduled track of a train. Formally, if $si = (n, arr, dep, act, t)$ then $si' = (n, arr, dep, act, t')$.

Stop-Skipping Stop Skipping allows a course to pass at a node where it would stop according to the original schedule. Formally, if $si = (n, arr, dep, Stop, t)$ then $si' = (n, arr, dep, Pass, t)$.

Partial Cancellation Partial Cancellation removes the start and/or end of a course. This is, if

$$S_c = (si_0, \dots, si_j, \dots, si_i, \dots, si_k)$$

then

$$S'_c = (\underbrace{\emptyset, \dots, \emptyset}_{j \text{ times}}, si_j, \dots, si_i, \underbrace{\emptyset, \dots, \emptyset}_{k-i \text{ times}}),$$

for some $0 \le j \le i \le k$.

Total Cancellation Total cancellation removes an entire course from the timetable. It is not operated at all and no node is served. So formally $S'_c = (\emptyset, \dots, \emptyset)$.

The timetable amendments above may disrupt the planned rolling stock duties. For example, the rolling stock of a partially canceled course is not available at the end node of the course to start operating the next course of its duty. Therefore, the amended timetable may also have amended rolling stock duties R'. R' is a set of sequences of duty elements. The following paragraph lists some exemplary possible amendments to R.

We may schedule empty rides between different nodes. This involves the addition of a new course $c' \notin C$ to the set C' with category $\text{ccat}'(c') = Empty$. Naturally, the added course has to follow the topology of the railway network. We may also send rolling stock units to a depot $n \in DEP$. This makes them available with type *Reserve* there. On the other hand, we can draw rolling stock with type *Reserve* from depots. Furthermore, we can adapt the duties to support short turning. Let $c_0, c_1 \in C$ be two consecutive courses of the same rolling stock duty. If both are partially canceled, such that c_0 ends at the same node n where the c_1 starts, we can short-turn the rolling stock. Additionally, $\text{st}(n) \neq \emptyset$ must hold. If we short-turn the rolling stock, we must also alter the start and end node of the corresponding change-end element in the rolling stock duty.

However, as it is not relevant to our approach (cf. Section 3.1.10), we do not further formalize the potential modifications of rolling stock duties.

2.4.5 Penalty computation

The objective function of the optimization problem is the economic penalty of the amended timetable. It should be minimized. The total penalty is defined as the sum of multiple independent components.

$$P_{total} = P_{ss} + P_{dd} + P_f$$

We define the components in the following sections.

Skipped stop penalty P_{ss}

Each planned stop that is skipped in the amended timetable causes an economic penalty. Stops that are not served because of a total or partial cancellation also count as skipped stops. The skipped stop penalty $P_{ss,c}$ of a single course $c \in C$ can be computed as follows:

$$P_{ss,c} = \sum_{si \in SS_c} \mathrm{bsv}(si) \cdot \mathrm{ssf}_c(si)$$

Here

$$\begin{aligned} SS_c := \{si \in S_c \mid\ & \mathrm{index}(si) \geq |RS_c| \\ & \wedge \mathrm{activity}(si) = \mathit{Stop} \\ & \wedge ((S'_c)_{\mathrm{index}(si)} = \emptyset) \vee \mathrm{activity}((S'_c)_{\mathrm{index}(si)}) = \mathit{Pass}\} \end{aligned}$$

is the set of schedule items, that are skipped in the amended timetable. However, only non-realized schedule items are considered. We define

$$bsv(si) := \mathrm{bsv}(\mathrm{node}(si), \mathrm{arrival}(si), \mathrm{cdir}(\mathrm{course}(si)))$$

Lastly, ssf_c is the so-called station stop factor. It is computed as follows: Let $\hat{SS_c}$ be the sequence containing the elements of SS_c in ascending order according to $\mathrm{bsv}(\cdot)$. Then

$$ssf_c(si) = \begin{cases} 35 & \text{if } si = (\hat{SS_c})_0 \\ 15 & \text{if } si = (\hat{SS_c})_1 \\ 1 & \text{otherwise} \end{cases} \tag{2.1}$$

As a consequence, the skipped stops with a lower base station value have a higher impact on the skipped stop penalty of a course.

Finally, the total skipped stop penalty is the sum of the penalties of all courses with type *Passenger*.

$$P_{ss} = \sum_{\substack{c \in C \\ \mathrm{ccat}(c) = \mathit{Passenger}}} P_{ss,c}$$

Destination delay penalty P_{dd}

Delays at the destination of a course also have an economic penalty. The destination of a course is the last node a course visits in the amended timetable. This is non-trivial in the case of partial cancellation. Courses that are totally canceled do not have a destination. Therefore, they do not impose a destination delay. Let si' be the last element of S'_c that is not $\emptyset$ if existent. Also, let $si = (S_c)_{\mathrm{index}(si')}$ be the corresponding element in the original schedule. Then, the destination delay is defined as $dd_c = \text{arrival}(si') - \text{arrival}(si)$. If si' is non-existent, $dd_c = 0$. The destination delay penalty of a single course c, $P_{dd,c}$, is defined as follows:

$$P_{dd,c} = \begin{cases} 0 & \text{if } dd_c < 180 \\ dd_c \cdot \frac{125}{60} & \text{if } dd_c \geq 180 \end{cases}$$

Note that a destination delay under 3 minutes does not impact the penalty. After that, the penalty scales linearly with the delay.

Finally, the total destination delay is the sum of the penalties of all courses with type *Passenger*.

$$P_{dd} = \sum_{\substack{c \in C \\ \text{ccat}(c) = \textit{Passenger}}} P_{dd,c} \tag{2.2}$$

Frequency penalty P_f

The frequency penalty penalizes amended timetables that have high headways between multiple arrivals in the same direction at certain reference nodes. The reference nodes are those defined in the set *FMS*. Additionally, there is a threshold for the maximum tolerated headway $h_t : \mathbb{N} \to \mathbb{N}$. It depends on the time of the day.

Let $r = (n, d) \in \textit{FMS}$ be a node direction tuple. Let

$$\begin{aligned} A_r = \{si \mid \exists c \in C : &\text{ccat}(c) = \textit{Passenger} \wedge \text{cdir}(c) = d \\ &\wedge si \in S'_c \wedge \text{node}(si) = n \\ &\wedge \text{activity}(si) = \textit{Stop} \wedge \text{arrival}(si) \neq -1\} \end{aligned}$$

be the set of all schedule items that entail an arrival at node n in direction d. Note that we only consider courses with category *Passenger*. Let $\hat{A}_r = (si_0, \ldots, si_{|A_r|-1})$ be the sequence containing all elements from A_r in ascending order based on arrival$(\cdot)$. Now, for every pair $p = (si_i, si_{i+1}), i < |A_r| - 1$, we define the headway $h_p = \text{arrival}(si_{i+1}) - \text{arrival}(si_i)$. We also define the headway threshold for the pair $h_t(p) = \max(h_t(\text{arrival}(si_i)), h_t(\text{arrival}(si_{i+1})))$. Then, the penalty for each pair is

$$P_{f,p} = \begin{cases} 0 & \text{if } h_p \leq h_t(p) \\ (h_p - h_t(p)) \cdot \frac{150}{60} & \text{if } h_p > h_t(p) \end{cases}$$

Again, it scales linearly after exceeding the threshold point. The total penalty for a node direction tuple $P_{f,r}$ is the sum of the penalties for all pairs that are not fully realized. More formally, given the snapshot time t we have

$$P_{f,r} = \sum_{\substack{0 \leq i < |A_r| - 1 \\ \text{arrival}(si_{i+1}) > t}} P_{f,(si_i, si_{i+1})}$$

Finally, the total frequency penalty is

$$P_f = \sum_{r \in \textit{FMS}} P_{f,r}$$

2.4.6 Hard constrains

The amended timetable must be consistent with the constraints implied by $\mathcal{I}$, $\mathcal{T}$ and $\mathcal{P}$. Those constraints are formalized in this section. Let $\mathcal{T}' = (C', \text{cdir}', \text{ccat}', (S'_c)_{c' \in C'}, R')$ be the amended timetable. Then, the following constraints apply:

Respecting the realized schedule $(RS_c)_{c\in C}$

Of course, any valid solution must be consistent with the realized schedule up to the snapshot time t. Let

$$c \in C, rsi \in RS_c, si = (S'_c)_{\text{index}(rsi)}$$

then the following must hold

$$\begin{aligned} & si \neq \emptyset \\ & \wedge \text{track}(rsi) = \text{track}(si) \\ & \wedge \text{node}(rsi) = \text{node}(si) \\ & \wedge \text{arrival}(rsi) = \text{arrival}(si) \\ & \wedge (\text{departure}(rsi) = \text{departure}(si) \vee \text{departure}(rsi) = -1) \end{aligned} \tag{2.3}$$

The formula models that schedule items that are already part of the realized schedule may not be modified in the amended schedule. The last clause of the conjunction respects the possibility that the departure of a realized schedule item was not realized before t. In this case, the departure in the amended timetable may differ from the departure in the realized schedule.

Another inconsistency with the realized schedule is the addition of more schedule items before the snapshot time. To avoid this, the following must hold: Let

$$c \in C', si \in (S'_c), i = \text{index}(si)$$

then

$$\begin{aligned} & (\text{arrival}(si) \geq t \wedge \text{departure}(si) \geq t) \\ & \vee |RS_c| > i \\ & \vee (\text{departure}(si) \geq t \wedge \text{departure}((RS_c)_i) = -1) \end{aligned}$$

Also, the realized schedule may be inconsistent with some of the following constraints. Therefore, the constraints only apply for schedule items whose departure time is after the snapshot time.

Dwell times

The amended timetable must respect the dwell times from the original timetable and the extended dwell times from the problem instance. Given that

$$c \in C, 0 < i < |S'_c| - 1, si' = (S'_c)_i \neq \emptyset, si = (S_c)_i, \text{activity}(si') = \mathit{Stop}$$

the following equations must hold. First

$$\text{departure}(si') - \text{arrival}(si') \geq \text{departure}(si) - \text{arrival}(si) \tag{2.4}$$

ensures that the dwell time in the amended timetable is at least as big as in the original timetable. Second,

$$\begin{aligned}\forall(s,e,n,dt)\in ESD : n &\neq \text{node}(si')\\ &\vee \text{departure}(si') \leq s\\ &\vee e + (\text{departure}(si) - \text{arrival}(si)) \leq \text{departure}(si')\\ &\vee (\text{departure}(si') - \text{arrival}(si')) \geq dt\end{aligned} \tag{2.5}$$

specifies that the extended dwell times at stations are respected. Note that in principle the extended dwell times apply whenever there is an intersection between the intervals (s, e) and $[\text{arrival}(si'), \text{departure}(si')]$. However, a course arriving shortly before the end of the incident may wait at the node until the incident is over. Then, it can dwell normally. As a consequence, it has a combined dwell time of the waiting time and the dwell time from the original timetable. This possibility is respected in the third clause of the disjunction of Formula 2.5. Finally, we have to consider the extended dwell time requirements of a single course. To enforce those, the following formula has to hold.

$$\begin{aligned}\forall(c^*,n,dt)\in ECD : c^* &\neq c\\ &\vee n \neq \text{node}(si')\\ &\vee (\text{departure}(si') - \text{arrival}(si')) \geq dt\end{aligned} \tag{2.6}$$

Runtimes

The amended timetable must respect the minimum runtimes mrt as well as the extended runtimes imposed by $ert \in ERT$. Let

$$c \in C', \emptyset \neq si_i \in (S'_c), si_{i+1} = \text{succ}(si_i) \neq \emptyset, l = (\text{node}(si_i), \text{node}(si_{i+1}))$$

Then, the minimum runtimes are enforced by

$$\text{arrival}(si_{i+1}) - \text{departure}(si_i) \geq \text{mrt}(l, \text{activity}(si_i), \text{activity}(si_{i+1})) \tag{2.7}$$

Furthermore, we enforce the extended runtimes by the constraint

$$\begin{aligned}\forall(s,e,l^*,rt)\in ERT : l^* &\neq l\\ &\vee \text{arrival}(si_{i+1}) \leq s\\ &\vee e + \text{mrt}(l, \text{activity}(si_i), \text{activity}(si_{i+1})) \leq \text{arrival}(si_{i+1})\\ &\vee \text{arrival}(si_{i+1}) - \text{departure}(si_i) \geq rt\end{aligned} \tag{2.8}$$

Note that again the extended runtime applies for all courses whose presence on the link has an intersection with the interval (s, e). However, a course may already enter a link before the incident is over. Then, If the incident is over before it could leave the link, its remaining runtime is assumed to be the default minimum runtime. This possibility is again respected by the third clause of the disjunction in Formula 2.8.

Minimum link headways

$\mathcal{T}'$ must respect the minimum headways given by lhw. This means that trains departing at the same node using the same link must be separated by the minimum headway time. Formally, let

$$\begin{aligned} c, c' \in C' \qquad & \emptyset \neq si_i \in S'_c & \emptyset \neq si_{i+1} = \mathrm{succ}(si_i) \\ & \emptyset \neq si_j \in S'_{c'} & \emptyset \neq si_{j+1} = \mathrm{succ}(si_j) \\ & \mathrm{node}(si_i) = \mathrm{node}(si_j) & \mathrm{node}(si_{i+1}) = \mathrm{node}(si_{j+1}) \end{aligned}$$

Then, the following must hold:

$$\begin{aligned} & (\mathrm{duty}(c) = \mathrm{duty}(c')) \\ \vee\, & \mathrm{departure}(si_j) - \mathrm{departure}(si_i) \\ & \quad \geq \mathrm{lhw}(\mathrm{activity}(si_i), \mathrm{activity}(si_{i+1}), \mathrm{activity}(si_j), \mathrm{activity}(si_{j+1}) \\ \vee\, & \mathrm{departure}(si_i) - \mathrm{departure}(si_j) \\ & \quad \geq \mathrm{lhw}(\mathrm{activity}(si_j), \mathrm{activity}(si_{j+1}), \mathrm{activity}(si_i), \mathrm{activity}(si_{i+1}) \end{aligned} \tag{2.9}$$

Also, trains may not overtake each other between stations. This is because each link models a single track. So trains departing first must arrive first. Formally

$$\begin{aligned} & \mathrm{departure}(si_j) < \mathrm{departure}(si_i) \implies \mathrm{arrival}(si_{j+1}) \leq \mathrm{arrival}(si_{i+1}) \\ \wedge\, & \mathrm{departure}(si_i) < \mathrm{departure}(si_j) \implies \mathrm{arrival}(si_{i+1}) \leq \mathrm{arrival}(si_{j+1}) \end{aligned} \tag{2.10}$$

Additionally, we have a special case for links with direction *Both*. Let the definitions be as before except that

$$\mathrm{node}(si_i) = \mathrm{node}(si_j + 1) \wedge \mathrm{node}(si_j) = \mathrm{node}(si_{i+1})$$

and

$$\mathrm{ldir}((\mathrm{node}(si_i), \mathrm{node}(si_{i+1}))) = \mathit{Both}$$

Then, the following must hold:

$$\begin{aligned} & (\mathrm{duty}(c) = \mathrm{duty}(c')) \\ \vee\, & \mathrm{departure}(si_j) - \mathrm{arrival}(si_{i+1}) \geq 30 \\ \vee\, & \mathrm{departure}(si_i) - \mathrm{arrival}(si_{j+1}) \geq 30 \end{aligned} \tag{2.11}$$

Minimum track headways

There is also a minimum headway between trains using the same track at the same node. This headway is 30 seconds and does not depend on any activity. So, given

$$\begin{aligned} c, c' \in C' \qquad & \emptyset \neq si_i \in S'_c & \emptyset \neq si_j \in S'_{c'} \\ & \mathrm{node}(si_i) = \mathrm{node}(si_j) & \mathrm{track}(si_i) = \mathrm{track}(si_j) \end{aligned}$$

we have

$$\begin{aligned} & (\mathrm{duty}(c) = \mathrm{duty}(c')) \\ \vee\, & \mathrm{departure}(si_j) - \mathrm{arrival}(si_i) \geq 30 \\ \vee\, & \mathrm{departure}(si_i) - \mathrm{arrival}(si_j) \geq 30 \end{aligned} \tag{2.12}$$

Note that here the difference is not between two departures or two arrivals but rather between a departure and an arrival. This is because a train occupies a track during its dwell time and the successor train may only arrive at the track after the first train already left. Also, we ignore track headway constraints between schedule items belonging to the same rolling stock duty. This is because the entire duty is served by the same train set. This train set cannot block itself.

Change end times

Furthermore, there needs to be a buffer between the same rolling stock operating two different courses. For a rolling stock duty $d \in R'$ this is modeled by the rolling stock duty elements $de \in d$ with $\text{type}(de) = \textit{ChangeEnd}$. For each $de \in d$ let $de_p \in d$ be closest element preceding de in d with $\text{type}(de_p) = \textit{Train}$. Symmetrically, let $de_s \in d$ be closest element succeeding de in d with $\text{type}(de_p) = \textit{Train}$. Also let $c_p = \text{course}(de_p)$ and $c_s = \text{course}(de_s)$. If those elements do not exist, the amended rolling stock duty is invalid. To meet the change end time constraint, each change end element de has to fulfill the following formula.

$$\text{endTime}(de) - \text{startTime}(de) \geq \text{cet}(\text{startNode}(de), \text{cdir}(c_p), \text{cdir}(c_s)) \tag{2.13}$$

However, sometimes the provided original timetable does not adhere to this constraint. Nonetheless, it should be a valid amended timetable. Therefore, we weaken Formula 2.13 if the change end element already existed in the original timetable. Let $de' \in d \in R$ be a change end duty element of an original rolling stock duty. Also let $\text{course}(de'_p) = c_p$ and $\text{course}(de'_s) = c_s$. Then, fulfilling the following formula suffices.

$$\text{endTime}(de) - \text{startTime}(de) \geq \text{endTime}(de') - \text{startTime}(de') \tag{2.14}$$

The formula implies that the amended change end time must be at least as big as the duration in the original timetable. Concluding, the duration of change end elements in the amended rolling stock duties must be at least as long as the minimum of the duration of the original change end element and the time specified by $\text{cet}(\cdot)$.

Rolling stock balance

When modifying the rolling stock duties, we need to adhere to a rolling stock balance. This means that the amount of rolling stock leaving a depot during the day must be equal to the amount entering it. This makes sure, that the timetable can be used for multiple days in a row without further adjustments. Therefore, for all $n \in N$ the following has to hold:

$$\sum_{\substack{d \in RS \\ \text{startNode}(d_0)=n}} 1 = \sum_{\substack{d \in RS \\ \text{endNode}(d_{|d|-1})=n}} 1$$

The equation makes sure that the number of duties starting at a node equals the number of duties ending there.

Late departures

As they are part of the given problem, $\mathcal{P}$, every $ld \in LD$ must also be respected by the amended timetable. Therefore, given that

$$(c, d) \in LD, si = (S_c)_0, si' = (S'_c)_0 \neq \emptyset$$

the following must hold

$$\text{departure}(si') - \text{departure}(si) \geq d \tag{2.15}$$

Early departure bound

Lastly, courses are not allowed to depart more than 5 minutes early compared to their departure in the original timetable. Formally, let

$$c \in C, 0 \leq i < |S'_c| - 1, si' = (S'_c)_i \neq \emptyset, si = (S_c)_i, \text{activity}(si') = \mathit{Stop}$$

Then, we have

$$\text{departure}(si) - \text{departure}(si') \leq 300 \tag{2.16}$$

Chapter 3

Approach

In this chapter, we present our approach for solving the problem of the 2022 RAS Problem Solving Competition. We propose an extension to the existing model introduced in Section 2.3.2 to make it applicable to the 2022 RAS problem (Section 3.1). Then, we introduce static implication rules for the extended model in Section 3.2. After that, Section 3.3 describes the concrete branch-and-bound algorithm that we use to find the optimal selection. Additionally, we discuss the management of metadata along the respective selections in the branch-and-bound algorithm from an implementation-centered perspective in Section 3.4. Lastly, we present two heuristics in Section 3.5 that reduce the size of the constructed alternative graphs.

3.1 Modeling the RAS problem using an alternative graph

In this section, we propose an alternative graph model that captures the complex constraints of the 2022 RAS problem. As a starting point, we extend the concept of an alternative graph in Section 3.1.1 to allow for more concise definitions in the remaining sections. Subsequently, in Sections 3.1.2 to 3.1.9, we introduce additional constraints and modifications to the existing model from Section 2.3.2. We describe the basics of our model in Section 3.1.2. Then, we address the possibility of skipping stops of individual courses in Section 3.1.3, consider variable runtimes on links in Section 3.1.4, and extend the model to express variable minimal headway times depending on the activities of trains in Section 3.1.5. Furthermore, we also consider the possibility to change tracks at nodes and its impact on track headway times in Section 3.1.6. Then, Section 3.1.7 shows how we can limit the early departure of courses while Section 3.1.8 focuses on the known future incidents. Finally, Section 3.1.9 explains how the model adheres to the realized schedule. To conclude, we quickly address some shortcomings of our proposed model in Section 3.1.10 and describe how we can extract the amended timetable $\mathcal{T}'$ from a complete consistent selection $\mathcal{S}$ in Section 3.1.11.

3.1.1 Extending alternative pairs

We extend the concept of an alternative graph $G = (V, F, A)$, by extending the notion of alternative pairs. We allow the alternatives to be a set of edges instead of a single edge. More formally, we let $A \subseteq 2^{V^2} \times 2^{V^2}$ instead of $A \subseteq V^2 \times V^2$. Let $p = (a_0, a_1) \in A$. Then, a_0 and a_1 are the two alternatives of the alternative pair p. Each $(h, k) \in a_i$ is an alternative edge of the alternative a_i for $i \in \{0, 1\}$. A selection $\mathcal{S}$ is still a set of edges. When selecting one alternative of a pair, we add all of its alternative edges to the selection. A selection may only be a superset of one alternative of each alternative pair in A. As a consequence, it may only contain the alternative edges of one of the alternatives. When used in the branch-and-bound

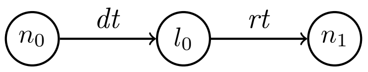


FIGURE 3.1: Connecting nodes and links using dwell and runtime

algorithm, this extension is equivalent to an original alternative graph with $e \leftrightsquigarrow f$ for all $e, f \in a_i$[1].

In the following, we also refer to an alternative pair as a *choice*.

3.1.2 Basic model

The 2022 RAS problem shares some similarities with the conflict resolution problem with fixed routes from Section 2.3. Those are, most notable, the scheduling of trains given a potentially divergent realized schedule. However, the CRFR problem requires microscopic detail on the given railway network. Meaning that we are given individual block sections of the network. Contrary to that, the infrastructure model $\mathcal{I}$ of the 2022 RAS problem is a macroscopic model. Therefore, we do not have access to individual block sections. However, we can reuse the existing alternative graph model to a certain extent.

Defining operations

Instead of considering individual block sections for defining operations, we can use less fine-grained resources. Every course $c \in C$ has to pass a sequence of nodes given by its schedule S_c. To achieve this, it also has to pass the links between the respective nodes. As a consequence, the nodes and links are the shared resources of the network. Therefore, we create an operation for every schedule item si of every course. Furthermore, we create operations for the links between nodes of the schedule items. As in the CRFR model, operations are modeled by vertices of the alternative graph. We connect those vertices using fixed edges with suitable weights that represent their processing time.

For instance, let $si \in S_c$ and $si' = \text{succ}(si) \neq \emptyset$. Then, we add three vertices n_0, n_1, l_0 to the alternative graph. They model two node operations (n_0, n_1) and a single link operation (l_0). In the following, when we refer to an operation, we always also mean its respective vertex. First, we connect the node operation to the next link operation using a fixed edge. The weight of the edge is the planned dwell time $dt = \text{departure}(si) - \text{arrival}(si)$ of the course at the first node . If either $\text{arrival}(si) = -1$ or $\text{departure}(si) = -1$ then $dt = 0$. After that, we add another fixed edge from the link operation to the second node operation. The weight of this edge is the minimum runtime for the link $l = (\text{node}(si), \text{node}(si'))$. This value depends on the activities at the link's start and end nodes. For this basic model, we use the predefined activities from the original schedule. Therefore, $rt = \text{mrt}(l, \text{activity}(si, \text{activity}(si')))$. Figure 3.1 depicts this example.

Defining jobs

Next, we have to group the operations into jobs. At first glance, having a job for each course sounds like a good approach. However, multiple courses are operated by the same rolling stock. Therefore, operations of different courses that belong to the same

[1] If $|a_0| \neq |a_1|$, the original alternative graph needs some additional dummy edges to be equivalent. Otherwise, assuming $|a_0| > |a_1|$, not every $e \in a_0$ has a partner $f \in a_1$ such that (e, f) is an alternative pair in the original graph

rolling stock duty cannot be executed concurrently. Consequently, each rolling stock duty $d \in R$ must have a single job. We already connected the operations within a course using fixed edges. Therefore, in the next step, we have to connect the different courses of the rolling stock duty.

Each rolling stock duty d is a sequence of rolling stock duty elements $(de_0, \dots, de_{n-1})$. Let $de_c, de'_c \in d$ be two duty element with type *Train*, such that de'_c is the first duty element succeeding de_c with type *Train*. Then, let $c = \text{course}(de_c)$ and $c' = \text{course}(de_{c'})$. Also let $de_{ce} \in d$ be the duty element with type *ChangeEnd* between de_c and de'_c. Instead of creating two different node operations for the last schedule item of S_c and the first of $S_{c'}$, we only create one. This is because between serving the first and the second course, the rolling stock does not leave the node. The processing time for this operation is the minimal required change end time cet. This is

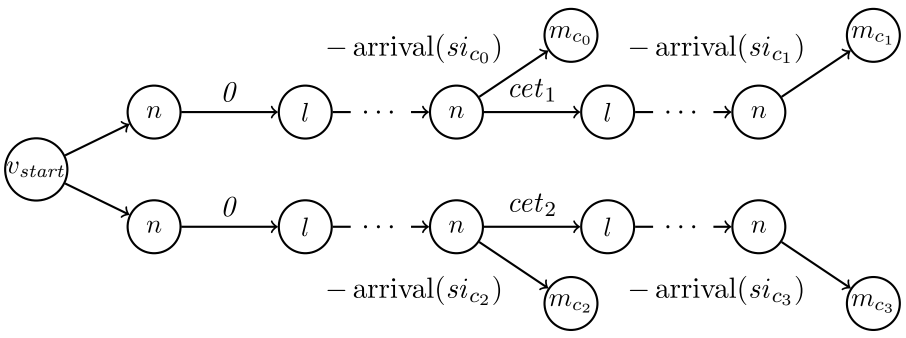
$$\begin{aligned} cet = \min(\, &\text{endTime}(de_{ce}) - \text{startTime}(de_{ce}), \\ &\text{cet}(\text{startNode}(de_{ce}), \text{cdir}(c), \text{cdir}(c'))) \end{aligned}$$

Now, for each rolling stock duty, let o_f be the first operation of the first course and o_l be the last operation of the last course. Then, $\text{lp}_\emptyset(o_f, o_l)$ is exactly the minimal time needed to complete the entire duty. When building the graph, we do not consider duty elements with type *Spare*. This is because they can be omitted in an amended timetable. They do not have any impact on the minimal time needed to complete a duty. Also, we do not consider reserve duties here. This is because their rolling stock is only sitting idle at a depot. It does not block any nodes or links for other trains.

As in the CRFR model, we add a dedicated start vertex v_{start}. From there, we add a fixed edge to the first operation of each rolling stock duty d. The weight of those edges is the planned start of the duty's first duty element. More formally $\text{startTime}((d)_0)$. Now, for each node operation o_n, $\text{lp}(v_{start}, o_n)$ is the earliest possible arrival time of the corresponding schedule item. For each link operation o_l, $\text{lp}(v_{start}, o_l)$ is the earliest possible departure time of the prior schedule item.

Extracting the objective function

Compared to the CRFR problem, the 2022 RAS problem does not consider the maximum secondary delay as its objective function. It rather has a more elaborate objective function that only partially depends on the delay of courses. Also, the term depending on delays does not consider the maximum delay, but rather the sum of individual delays (cf. Equation 2.2). Therefore, adding fixed edges to a single dedicated end vertex for measuring the objective function is not sufficient. However, we still want to extract the delay term of the objective function from one or multiple longest paths in the graph $G(\mathcal{S})$. To achieve this, we add a set of measure vertices M to the vertices of our alternative graph. M contains one vertex for every course $c \in C$ with $\text{ccat}(c) = \textit{Passenger}$. We refer to this vertex as m_c in the following. Now, for every $c \in C$, we add a fixed edge from its last node operation to m_c. The weight of this edge is the negative arrival time of the last schedule item of S_c. More formally, $-\text{arrival}((S_c)_{|S_c|-1})$. As a consequence, $\text{lp}_\mathcal{S}(m_c)$ is equal to the destination delay dd_c of c. In conclusion, we can extract the delay penalty $P_{dd,c}$ of individual courses and the total delay penalty P_{dd} from the longest paths from the start vertex to the measure vertices. The big difference to the prior model is that we need to evaluate multiple longest paths instead of a single one.



v_{start}
n
0
l
n
$- \text{arrival}(si_{c_0})$
m_{c_0}
cet_1
l
n
$- \text{arrival}(si_{c_1})$
m_{c_1}
n
0
l
n
cet_2
l
n
$- \text{arrival}(si_{c_2})$
m_{c_2}
$- \text{arrival}(si_{c_3})$
m_{c_3}

FIGURE 3.2: Graph with a start vertex, measure vertices and change end times

Figure 3.2 shows a simplified example graph for two rolling stock duties. Each of the duties contains two passenger courses. It contains four measure nodes and depicts the change end time between courses. Intermediate operations of courses are left out for clarity.

So far, the model presented does not contain any alternative pairs. Furthermore, it does not model any interactions between different duties. Therefore, finding an optimal complete selection is trivial. However, in the following sections, we extend it with alternative pairs to model the additional constraints and freedoms of the 2022 RAS problem.

3.1.3 Stop/Pass choices

The 2022 RAS problem allows us the skip stops in the amended timetable. We model this in the alternative graph using alternative edges. We add an alternative pair $p = (a_{Pass}, a_{Stop})$ to A for every schedule item si from the original schedule with $\text{activity}(si) = \mathit{Stop}$. The semantics of the alternative a_{Pass} is that the stop is skipped in the amended schedule. The semantics of the alternative a_{Stop} is that the course stops at the respective node. We now need to find appropriate edges to add to a_{Pass} and a_{Stop} such that they model the defined semantics in the graph. Apart from that, we want to make sure that, independently of the selection $\mathcal{S}$, the longest path to any measure vertex m_c is a lower bound for the destination delay dd_c. This especially includes selections that contain neither a_{Pass} nor a_{Stop}.

Let n be a node operation and l the following link operation. Let dt be the dwell time of the corresponding schedule item. Instead of having a fixed edge (n, l) with weight dt, we now add multiple edges. First, one alternative edge $\alpha_0 = (n, l) \in a_{Stop}$ with weight $w(\alpha_0) = dt$. Second, one alternative edge $\alpha_1 = (n, l) \in a_{Pass}$ with weight $w(\alpha_1) = 0$. The weights model the time a train stays at the node when it stops $w(\alpha_0)$ or passes $w(\alpha_1)$. Now, whenever a selection neither contains a_{Pass} nor a_{Stop}, the succeeding operations of the rolling stock duties are no longer reachable from v_{start}. To remedy this, we need to add a fixed edge $f = (n, l) \in F$ with $w(f) = 0$ to the graph. This fixed edge is a lower bound for the two alternative edges α_0 and α_1. As f has the same weight, target and origin as α_1, we can simplify the graph by removing α_1 again. Figure 3.3 shows the resulting graph.

In the following sections, we refer to the alternative pairs added in this way as *stop/pass* alternative pairs.

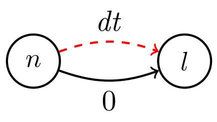


FIGURE 3.3: Modeling stop/pass alternative pairs with $\{(\emptyset, 🟥)\} \subseteq A$

3.1.4 Variable runtimes

In the 2022 RAS problem, the minimum runtime on links depends on the activities at the link's start and end nodes (cf. Equation 2.7). Since choosing those activities is part of the optimization problem, the minimum link runtime is unknown beforehand. Therefore, we cannot model the runtime with a single fixed edge with a predetermined weight. We rather have to introduce a combination of alternative edges that model the different potential runtimes. To achieve this, we re-use the stop/pass alternative pairs we created as part of Section 3.1.3. We do that as their semantics correspond to the activities of courses. Again, our goal is to make sure that the length of the longest paths to measure nodes corresponds to a lower bound for the destination delay. We also want to make this lower bound as strict as possible for partial selections.

We cannot encode the duration of a link in a single edge, as it depends on two choices. To remedy this, we introduce additional vertices to the graph. Those vertices are guarded by alternative edges with weight 0. The edges are part of one alternative of the first choice. The new vertices are only reachable if the corresponding alternative is selected. As a consequence, we can assume this *implicit state* when setting the weight of outgoing edges from this vertex. Using this technique, we can encode durations of operations that depend on multiple choices.

In our concrete case, let si_0, si_1 be two schedule items. Let n_0, n_1, l_0, l_1 be their corresponding node and link operations. Let $(a_{0,Pass}, a_{0,Stop})$ and $(a_{1,Pass}, a_{1,Stop})$ be the stop/pass alternative pairs of the node operations. Also, let $\ell = (\text{node}(si_0), \text{node}(si_0)) \in L$ be the link between the nodes of the schedule items. We introduce two new vertices s_0 and p_0. s_0 shall encode the information that the course stopped at the first node. On the other hand, p_0 encodes that the course passed the node. Therefore, we add the alternative edges $(l_0, p_0) \in a_{0,Pass}$ and $(l_0, s_0) \in a_{0,Stop}$. Both have weight 0. Then, we add the alternative edges listed in the following table

edge	alternative	weight
(p_0, n_1)	$a_{1,Pass}$	$\text{mrt}(\ell, PASS, PASS)$
(p_0, n_1)	$a_{1,Stop}$	$\text{mrt}(\ell, PASS, STOP)$
(s_0, n_1)	$a_{1,Pass}$	$\text{mrt}(\ell, STOP, PASS)$
(s_0, n_1)	$a_{1,Stop}$	$\text{mrt}(\ell, STOP, STOP)$

This construction models the runtime correctly given that both stop/pass alternative pairs are selected. However, we do not have any lower bounds given that none or only one of the alternative pairs is selected. Therefore, we introduce further alternative and fixed edges that handle those cases. They are listed in the following table

edge	alternative	weight
(p_0, n_1)	-	$\min_{a_1 \in Activities}(\text{mrt}(\ell, PASS, a_1))$
(s_0, n_1)	-	$\min_{a_1 \in Activities}(\text{mrt}(\ell, STOP, a_1))$
(n_0, n_1)	-	$\min_{a_0, a_1 \in Activities}(\text{mrt}(\ell, a_0, a_1))$
(n_0, n_1)	$a_{1,Pass}$	$\min_{a_0 \in Activities}(\text{mrt}(\ell, a_0, PASS))$
(n_0, n_1)	$a_{1,Stop}$	$\min_{a_0 \in Activities}(\text{mrt}(\ell, a_0, STOP))$

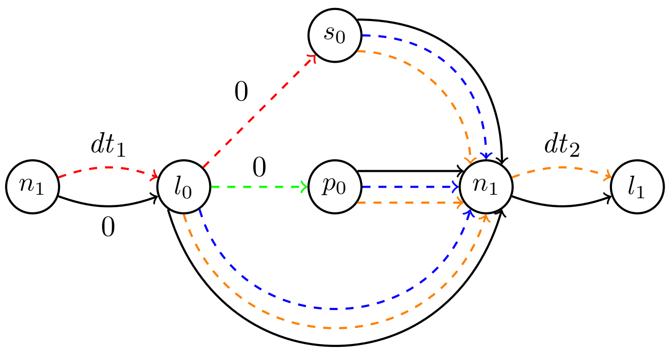


FIGURE 3.4: Adding variable minimum runtimes to the model with $\{($🟩, 🟥$)($🟦, 🟧$)\} \subseteq A$

Their respective weights are the minimum potential runtime, given all decisions implicitly present in their origin node. Figure 3.4 shows this example graphically.

For further simplification, it is possible to remove those alternative edges from the graph for which there is a fixed edge with the same weight, origin, and target. This is similar to the case in Section 3.1.3. Also, the construction can be simplified if there is no stop/pass choice for either the source or target schedule item. This is the case if the course already passes at one of the nodes in the original schedule. If none of the two choices exists, then a simple fixed edge suffices. If only one choice exists, the runtime can be modeled similarly to the dwell time in the last section.

3.1.5 Variable link headways

The 2022 RAS problem has a constraint for the minimum headway between two trains entering the same link. Additionally, trains may not overtake each other in a link (cf. Equations 2.9 and 2.10). This constraint is different from the CRFR problem, as multiple trains can use the same link at the same time, as long as they adhere to the minimum headways. Additionally, the headways depend on the activities at the respective source and target nodes. This no longer corresponds to the exclusive usage of machines in the job-shop scheduling problem. Therefore, we must adapt our model. However, we still determine the order in which trains enter links using the alternative edges introduced by these constraints.

First, we assume that the link headway is fixed and does not depend on the surrounding activities. Let n_0, l_0, n_1 be a sequence of node and link operations from course $c_0 \in C$. Let n_2, l_1, n_3 be a sequence of node and link operations from course $c_1 \in C$ that corresponds to the same nodes and link. Let $\ell \in L$ be this link. We introduce a new alternative pair $p_{l_0,l_1} = (a_0, a_1) \in A$ with the following semantics. Alternative a_0 represents that course c_0 enters link ℓ first. a_1 models that c_1 enters ℓ first. Let hw_0 be the necessary headway needed between the two courses if c_0 enters the link first. Symmetrically, let hw_1 be the necessary headway if c_1 enters first. Then, we add the edge (l_0, l_1) with weight hw_0 to a_0 and the edge (l_1, l_0) with weight hw_1 to a_1. These edges suffice to model the minimum required headway. However, they still allow for overtaking. To avoid this, we further add the alternative edges (n_1, n_3) and (n_3, n_1) to a_0 and a_1 respectively. Both edges have weight 0 since the 2022 RAS problem does not forbid courses to enter a node at the same time. We call the alternative pairs created this way *link headway* alternative pairs. Figure 3.5 shows the construction described above.

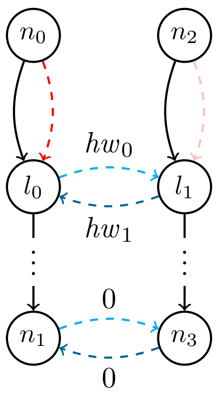


FIGURE 3.5: Fixed headway constraint between two link operations with (🟦, 🟦) $\in A$

The introduction of dependent headway times makes the construction more complicated. As $\mathrm{lhw}(\ell,\cdot,\cdot,\cdot,\cdot)$ depends on 4 activities, we potentially have 16 different headway values. Furthermore, we have to make sure that each headway value is only considered if the respective stop/pass alternative pairs are selected. Additionally, we also want good lower bounds if only a subset of the alternative pairs is selected. We again make use of guarded vertices to realize this.

Let $n_0, l_0, n_1, n_2, l_1, n_3$ be as in the last paragraph. Let $p_i = (a_{i,p}, a_{i,s}) \in A$ be the stop/pass alternative pair of node operation n_i for $i \in \{0,1,2,3\}$. Also, let $p_{l_0,l_1} = (a_0, a_1) \in A$ be the alternative pair modeling which course enters ℓ first. To start, we only model the possibility that c_0 enters the link first. The other possibility can be modeled symmetrically. First, we create the following guarded vertices with the intended implicit states

vertex	intended implicit state
px_0	c_0 passes at n_0
sx_0	c_0 stops at n_0
pp_0	c_0 passes at n_0 and stops at n_1
ps_0	c_0 passes at n_0 and passes atn_1
sp_0	c_0 stops at n_0 and stops at n_1
ss_0	c_0 stops at n_0 and passes atn_1

We connect them to the remaining graph from l_0 using the following 0-weighted alternative edges

$$(l_0, px_0) \in a_{0,p} \qquad (l_0, sx_0) \in a_{0,s}$$
$$(px_0, pp_0) \in a_{1,p} \qquad (px_0, ps_0) \in a_{1,s}$$
$$(sx_0, sp_0) \in a_{1,p} \qquad (sx_0, ss_0) \in a_{1,s}$$

Similarly, we create the vertices $px_1, \dots, ss_1$. However, they are not guarded in the sense that they are only reachable if certain alternatives are selected. Rather, we make sure that the remaining graph can only be reached from those vertices if the corresponding alternatives are selected. To accomplish this, we create the same edges as we did for $px_0, \dots, ss_0$ but in the reverse direction. They are part of the alternatives of p_2 and p_3. For example, we create the edges $(px_1, l_1) \in a_{2,p}$ and $(pp_1, px_1) \in a_{3,p}$. Finally, we create the edges $((y_0y_1)_0, (y_2y_3)_0) \in a_0$ for $y_i \in \{p, s\}$. These 16 edges fully connect the vertices with a full implicit state. Their

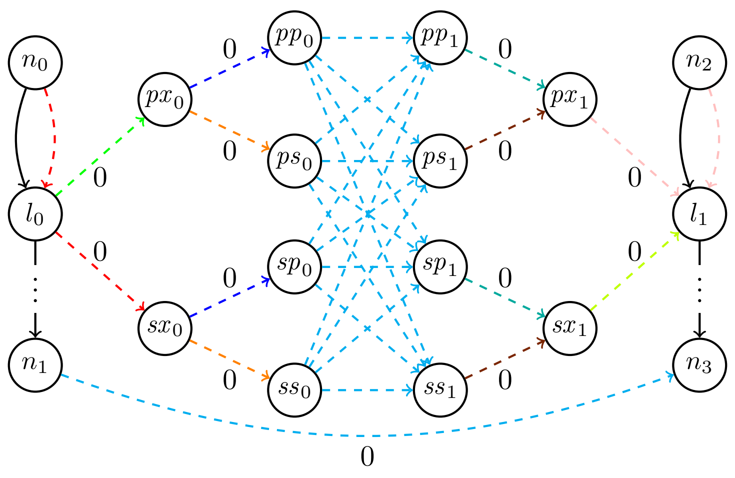


FIGURE 3.6: Basic variable link headway constraint with {(🟩, 🟥)(🟦, 🟧), (🟪, 🟩), (🟩, 🟫), (🟦, 🟦)} $\subseteq A$

corresponding weights depend on these full implicit states. For example, the weight of (sp_0, ps_s) is $LHW(\ell, \mathit{Stop}, \mathit{Pass}, \mathit{Pass}, \mathit{Stop})$. Now, we have a path from l_0 to l_1 with length $\text{lhw}(\ell, ac_0, ac_1, ac_2, ac_3)$ iff the activities of the courses at nodes n_i is ac_i and course c_0 enters the link first. Furthermore, we keep the edge $(n_1, n_3) \in a_0$ to prevent overtakes. Figure 3.6 depicts this. As already noted above, we can cover the case when c_1 enters the link first by a symmetrical construction.

Better Lower Bounds

The construction described above is functionally correct. However, it does not achieve our goal of producing good lower bounds. This is because there is no path between the two link operations if not all stop/pass alternative pairs are selected. To remedy this, we introduce additional alternative edges. The most simple addition is the edge $(l_0, l_1) \in a_0$ with weight

$$\min_{ac_i \in \mathit{Activities}} (\text{lhw}(\ell, ac_0, ac_1, ac_2, ac_3))$$

With the added edge, we get a lower bound for the headway if no stop/pass alternative pair is selected. We can further improve this by adding alternative edges for every combination of vertices. For this, we first introduce the vertices xp_i and xs_i for $i \in \{0, 1\}$. They have the intended implicit state that c_i stops or respectively passes at the link's target node. The state has no information about the activity at the source node. We connect them directly to l_i using appropriate alternative edges. Then, we define the two sets

$$HW_{l_0,out} = \{l_0, xp_0, xs_0, px_0, sx_0, pp_0, ps_0, sp_0, ss_0\}$$
$$HW_{l_1,in} = \{l_0, xp_0, xs_0, px_0, sx_0, pp_0, ps_0, sp_0, ss_0\}$$

We call the vertices in those sets headway vertices. They are all the vertices that might need to be connected by alternative edges with appropriate weight to achieve

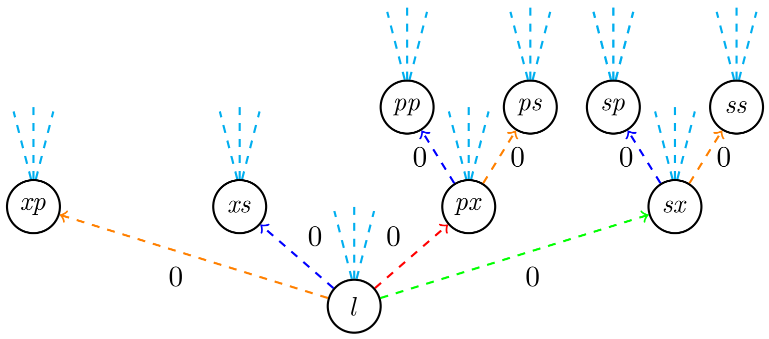


FIGURE 3.7: Full Variable headway constraint with $p_{l_0,l_1} = \{$🟦, 🟦$\}$

the best possible lower bounds. We define the two functions

$$\begin{aligned} \text{startAct} &: HW \to \mathit{Activities} \cup \{\mathit{Unknown}\} \\ \text{endAct} &: HW \to \mathit{Activities} \cup \{\mathit{Unknown}\} \end{aligned}$$

They are defined such that they extract the implicit state of headway vertices. For example, $\text{startAct}(sx_0) = \mathit{Stop}$ and $\text{endAct}(sx_0) = \mathit{Pass}$. Furthermore, we define the function $\widetilde{\text{lhw}} : L \times (\mathit{Activities} \cup \{\mathit{Unknown}\})^4 \to \mathbb{N}$ with

$$\widetilde{\text{lhw}}(\ell, \widetilde{ac}_0, \widetilde{ac}_1, \widetilde{ac}_2, \widetilde{ac}_3) = \min_{\substack{ac_i \in \mathit{Activities} \\ ac_i = \widetilde{ac}_i \\ \vee \widetilde{ac}_i = \mathit{Unknown}}} (\text{lhw}(\ell, ac_0, ac_1, ac_2, ac_3))$$

$\widetilde{\text{lhw}}$ gives a lower bound for the link headway given that only a subset of the activities is known. Now, for every $v_0 \in HW_{l_0,out}, v_1 \in HW_{l_1,in}$ we add the edge (v_0, v_1) to a_0. It has the weight

$$w((v_0, v_1)) = \widetilde{\text{lhw}}(\text{startAct}(v_0), \text{endAct}(v_0), \text{startAct}(v_1), \text{endAct}(v_1))$$

As a consequence, the longest path from l_0 to l_1 is always the best possible lower bound given the already selected alternative pairs. Figure 3.7 shows parts of the construction schematically.

As in the last section, we create some redundant edges if we connect every pair v_0, v_1. For instance, let us consider the edges $(pp, v_1), (ps, v_1), (px, v_1)$ for some $v_1 \in HW_{l_1,in}$. Then, $w((px, v_1)) = w((pp, v_1)) \vee w((px, v_1)) = w((ps, v_1))$ by the definition of $\widetilde{\text{lhw}}$. Therefore, either (pp, v_1) or (ps, v_1) can be removed from the graph. In general, we can remove any edge for which there exists a less specific edge, with the same weight. More formally, we can remove an edge $(v_0, v_1) \in HW_{l_0,out} \times HW_{l_1,in}$ if the following formula holds.

$$
\begin{aligned}
\exists v_0' \in HW_{l_0,out} :& \operatorname{startAct}(v_0') = \mathit{Unknown} \land \operatorname{endAct}(v_0') = \operatorname{endAct}(v_0) \\
& \land v_0' \neq v_0 \land w((v_0', v_1)) = w((v_0, v_1)) \\
\lor \exists v_0' \in HW_{l_0,out} :& \operatorname{endAct}(v_0') = \mathit{Unknown} \land \operatorname{startAct}(v_0') = \operatorname{startAct}(v_0) \\
& \land v_0' \neq v_0 \land w((v_0', v_1)) = w((v_0, v_1)) \\
\lor \exists v_1' \in HW_{l_1,in} :& \operatorname{startAct}(v_1') = \mathit{Unknown} \land \operatorname{endAct}(v_1') = \operatorname{endAct}(v_1) \\
& \land v_1' \neq v_1 \land w((v_0, v_1')) = w((v_0, v_1)) \\
\lor \exists v_1' \in HW_{l_1,in} :& \operatorname{endAct}(v_1') = \mathit{Unknown} \land \operatorname{startAct}(v_1') = \operatorname{startAct}(v_1) \\
& \land v_1' \neq v_1 \land w((v_0, v_1')) = w((v_0, v_1))
\end{aligned}
$$

Furthermore, we can simplify the sets $HW_{l,d}$ ($d \in \{out, in\}$) if not all stop/pass alternative pairs exist. For example, if there is neither a stop/pass alternative pair at the prior nor at the following node then $HW_{l,d} = \{l\}$. If there is only a stop/pass alternative pair at the prior node and the activity at the following node is fixed then $HW_{l,d} = \{l, sy, py\}$. Here $y \in \{s, p\}$ according to the fixed activity. Symmetrically, $HW_{l,d} = \{l, ys, yp\}$ if it is the other way around. In both cases, we also adapt $\operatorname{startAct}(\cdot)$ and $\operatorname{endAct}(\cdot)$. Then, the functions also consider the predefined activities. We connect the remaining vertices in $HW_{l,d}$ using appropriate guarding edges.

Finally, we can reuse the sets $HW_{l,d}$ for different headway constraints. For example, let l_2 be another link operation that utilized the same link as l_0. Then, we can reuse the nodes in $HW_{l_0,out}$ and $HW_{l_0,in}$. We simply have to create a new alternative pair $p_{l_0,l_2} = (a_2, a_3)$. To complete the construction, we add the alternative edges connecting $HW_{l_0,out}$ with $HW_{l_2,in}$ to a_2 and edges connecting $HW_{l_2,out}$ with $HW_{l_1,in}$ to a_3. For the consistency of our function definitions, we define HW to be the union of all $HW_{l,d}$.

3.1.6 Track headways

After investigating link headway constraints in the last section, we now focus on track headway constraints. They specify that a track at a node must stay clear for at least 30 seconds before another course may occupy it (cf. Equation 2.12). This is a simple task if the different nodes only have a single track or if the tracks used by the courses are predetermined. However, it becomes more challenging once the used tracks can be changed dynamically. The latter is the case for the 2022 RAS problem. It allows for re-platforming trains at nodes. Nonetheless, for a better introduction to the more elaborate case, we start with the scenario where dynamic track changes are not possible.

Single-Track Case

Let $c_0, c_1 \in C$ be two courses that both need to visit a node $n \in N$. Let n_0, n_1 be the corresponding node operations of c_0, c_1 for n. Let $d_i = \operatorname{cdir}(c_i)$ be the direction of c_i for $i \in \{0, 1\}$. We assume for now that both courses use the same track at the node. There could be several reasons for that. Either because the node has only one usable track. This is the case if $\operatorname{tracks}(n, d_0) = \operatorname{tracks}(n, d_1)$ and $|\operatorname{tracks}(n, d_0)| = 1$. Another possibility is that we want to forbid re-platforming for some operations and the tracks of the corresponding schedule items in the original schedules S_{c_0} and S_{c_1}

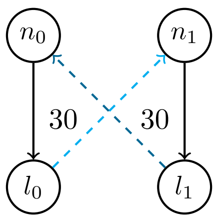


FIGURE 3.8: Track headway constraint at a node with a single track.
(🟦, 🟦) $\in A$

are the same. In any case, then the track behaves similarly to a machine in the job-shop scheduling problem. Both operations need to occupy this machine. The machine has a fixed setup time of 30 seconds. For modeling, we introduce the alternative pair $p_{n_0,n_1} = (a_0, a_1)$. Here, the selection of a_0 has the semantics that c_0 enters the node n first. Symmetrically, the semantics of a_1 is that c_1 enters n first. We can model the fixed setup time by two alternative edges $(l_0, n_1) \in a_0$ and $(l_1, n_0) \in a_1$. Both have weight 30. We assumed that $l_i = \sigma(n_i)$ for $i \in \{0, 1\}$ are the successor operations of the node operations. Figure 3.8 depicts this construction. We call the alternative pairs created this way *track headway* alternative pairs.

Multi-Track Case

If there are multiple available tracks at a node, we cannot model this constraint using a single alternative pair. In this case, two trains may occupy two different tracks of the same node at the same time without violating any constraints. One train may even overtake another if there is more than one track available. Also, we have to incorporate re-platformings in our model.

We start with creating alternative pairs to model re-platformings. Creating a single alternative pair for modeling the track choice of a course at a node does not suffice in every case. This is because there are only two alternatives in an alternative pair. However, there might be more than two available tracks. Therefore, we have to encode the track to be chosen into multiple alternative pairs. To do this, we first define some helper functions. Let n_i, c_i, d_i be as in the single track scenario. Let T_{n_i} be the set of available tracks for operation n_i. In the default case $T_{n_i} = \text{tracks}(n, d_i)$. We define $\text{order}_{n_i} : T_{n_i} \to \mathbb{N}$ with

$$\text{order}_{n_i}(t) = i \text{ iff } t \text{ is the } (i+1)\text{-th smallest element of } T_{n_i}$$

For instance, if $T_{n_i} = \{0, 2, 5\}$ then $\text{order}_{n_i}(5) = 2$. We do this, to get continuous identifiers for the available tracks for a course at a node. Now, we create $m = \lceil \log_2(|T_{n_i}|) \rceil$ alternative pairs $p_{i,0} = (a_{i,0,0}, a_{i,0,1}) \ldots p_{i,m-1} = (a_{i,m-1,0}, a_{i,m-1,1}) \in A$. Each alternative $a_{i,j,k}$ has the semantics that the $(j-1)$-th least significant bit of $\text{order}_{n_i}(t)$ is k. Using this technique, we can express the selection of the used track using binary alternatives. We call the alternative pairs created this way *track* alternative pairs.

For a better understanding, we provide an example. We assume that the node n has three tracks. One for the westbound direction and two that can be used in both directions. Also, assume that $d_0 = WB$ and $d_1 = EB$. Then, $T_{n_0} = \{0, 1, 2\}$ and $T_{n_1} = \{1, 2\}$. In this special case $\text{order}_{n_0}(t) = t$. As $\lceil \log_2(|T_{n_1}|) \rceil = 2$ we create two alternative pairs $(a_{0,0,0}, a_{0,0,1})$ and $(a_{0,1,0}, a_{0,1,1})$. If we add $a_{0,0,0}$ and $a_{0,1,1}$ to a selection, then this has the following semantics. Course c_0 occupies the track t at node n for which $\text{order}_{n_0}(t) = 2 = 10_2$. In this case $t = 2$. The situation is more

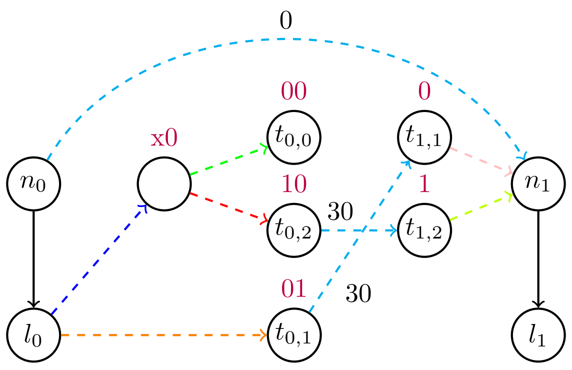


FIGURE 3.9: Track headway constraint at a node with a multiple tracks. $\{p_{0,1} = (🟩, 🟥), p_{0,0} = (🟦, 🟧), p_{1,0} = (🟥, 🟩), p_{n_0,n_1} = (🟦, 🟦)\} \subseteq A$

complicated if we select $a_{0,0,1}$ and $a_{0,1,1}$. Then, $\text{order}_{n_0}(t) \overset{!}{=} 3 = 11_2$. As $|T_{n_0}| = 3$, there does not exist such a track. In this case, we ignore the alternative corresponding to the most significant bit. As there is no track with $\text{order}_{n_0}(t) = 3$ it is redundant given the other alternative. As a consequence, the occupied track is the one with $\text{order}_{n_0}(t) = 1 = 01_2$. For c_1 we only need a single alternative pair to decide which track it uses. However, here $\text{order}_{n_1}(t) \neq t$.

In the second step, we use the track alternatives for modeling the track headway constraints. We use a similar construction as in Section 3.1.5. This means that we want to guard the alternative edges causing the actual headway. We want to make sure that there can only be a path between l_0 and n_1 or l_1 and n_0 respectively if both courses occupy the same track.

In the following, we restrict the model to the case that c_0 enters the node first as the reverse can be handled symmetrically. We create a vertex $t_{0,i}$ for every element $i \in T_{n_0}$. Those vertices shall have the implicit state that c_0 uses the track i. Therefore, we connect l_0 to the new vertices using guarding edges. If there are more than two tracks available, this also requires intermediate vertices with a partial implicit state. We need at most as many edges as there are track alternative pairs. However, instead of connecting l_0 to each of those vertices with a disjunct path, we can build a binary tree. This reduces the number of edges needed. The root of the binary tree is l_0. The first branching decides the least significant bit of $\text{order}_{n_0}(i)$. The second branching decides the second least significant bit up until the $\lceil \log_2(|T_{n_0}|) \rceil$'th branching decides the most significant bit. We add the alternative edges of the tree to the alternative of the respective track alternative pairs. If a track is completely specified without the most significant bit, we leave out the last branching. Then, we create the vertices $t_{1,j}$ for $j \in T_{n_1}$. We also connect them using a binary tree. However, we change the root node to n_1. Furthermore, we change the direction of the edges, such that there are paths from the leaves to the root. Let $p_{n_0,n_1} = (a_0, a_1)$ be defined with the same semantics as in the single-track scenario. To complete the construction, we connect $t_{0,i}$ to $t_{1,i}$ for $i \in T_{n_0} \cap T_{n_1}$ using alternative edges with weight 30. Those edges are part of a_0. The edges created by a symmetric construction are part of a_1. Figure 3.9 shows the construction for the concrete example.

Furthermore, Figure 3.9 contains the alternative edge $(n_0, n_1) \in a_0$ with weight 0. It is necessary for two reasons. First, we want to keep the original semantics of alternative pair p_{n_0,n_1}. This means that the selection of a_0 has the semantics that c_0 enters the node first. However, if c_0 and c_1 use different tracks, all other edges of a_0 are not reachable. Therefore, they do not ensure the defined semantics of the alternative. Moreover, the additional edge achieves better lower bounds. Without it p_{n_0,n_1} has no impact on any longest path if we have not decided which tracks are used by c_0 and c_1. This is bad for the branch-and-bound approach, as otherwise we cannot estimate the consequence of the decision directly. For completeness, we also add the edge (n_1, n_0) to a_1. Those edges are not necessary, if both courses are forced to use the same track.

As in Section 3.1.5, we can reuse some of the created vertices and edges for other track headway constraints. We only have to create the outgoing and incoming binary trees once. We connect their leave notes according to the different headway constraints. Also, if there is no intersection between T_{n_0} and T_{n_1} we do not have to create an alternative pair at all.

3.1.7 Limiting early departures

The 2022 RAS problem does not allow courses to depart more than 5 minutes early (cf. Equation 2.16). We can easily encode this constraint in our model by adding additional alternative edges. Let n be a node operation and l the succeeding link operation. We can restrict the earliest possible departure at the node by an edge $e = (v_{start}, l)$. Then, the course may not enter the link before the time that is equal to the weight $w(e)$. Concluding, we set $w(e) = \text{departure}(si) - 300$. Here we assume that si is the schedule item that caused the creation of n in the first place.

Early departures are only limited at nodes where courses stop. Therefore, if there is a stop/pass alternative pair for the course at n, we make the edge e an alternative edge. We add it to the alternative of the respective pair with the semantics that the course stops at the node. In the other case, activity at the node is predetermined. Then, we add a fixed edge if the activity is *Stop*, and no edge if it is *Pass*.

3.1.8 Respecting future incidents

The 2022 RAS problem contains known future incidents with four different types (cf. Section 2.4.3). We have to respect all of them in our proposed model.

First, our amended timetable has to consider late departures (cf. Equation 2.15). However, we can extend the model easily to respect those additional constraints. Let $ld = (c, d) \in LD$. Also, let l be the first link operation of the course c. Then, we add the fixed edge $e = (v_{start}, l)$ with weight $w(e) = \text{departure}((S_c)_0) + d$ to the alternative graph. This simple addition fully covers the constraint.

The second type of known future incidents is an extended course dwell time $ecd = (c, n, dt) \in ECD$. They specify that a course dwells longer than planned at a station (cf. Equation 2.6). To model this, we simply adapt the dwell time we consider when connecting the respective node and link operations to dt. Additionally, we force courses to stop at the respective node by removing any potential stop/pass alternative pairs.

Incorporating the last two incident types ($esd \in ESD$ and $ert \in ERT$) in our model is more complicated. This is because both incidents only have an effect within a certain period of time. This means that courses passing the respective location of the incident outside of this period are not affected. We cannot express this using the primitives we introduced so far. If we are only using edges with a predefined weight, the processing time of individual operations cannot depend on their start time. Therefore, we introduce time-dependent edges. This means that we extend the weight function to additionally depend on a point in time. Then, $w(e, t) = x$ expresses that the weight of edge e at time t is equal to x. When computing longest paths, the

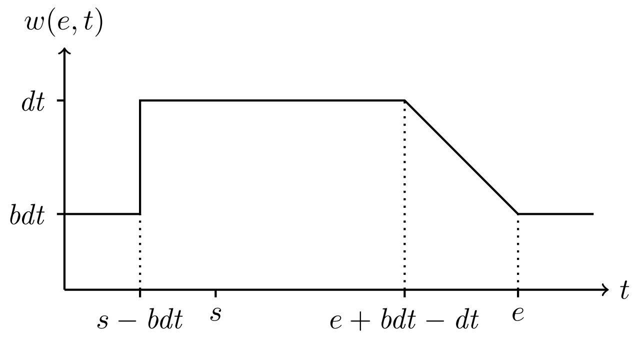


FIGURE 3.10: Plotted extended dwell time

used time t is the length of the longest path from the start vertex to the origin of the edge. Also, as most edges are not time-dependent, we still write $w(e) = x$. We define it to be equivalent to $\forall t \in \mathbb{N} : w(e,t) = x$.

Now, let $esd = (s, e, n, dt) \in ESD$. For any course c having a node operation for node n let $e_{c,n}$ be the edge with the weight of the course's dwell time at the node. Let bdt be this base dwell time without any modifications. $e_{c,n}$ may be an alternative or fixed edge depending on whether there is a stop/pass alternative pair at the respective node operation. In any case, we define its new weight as follows:

$$w(e_{c,n}, t) = \begin{cases} bdt & \text{if } t \leq s - bdt \\ edt & \text{if } t < e + bdt - dt \\ e + bdt - t & \text{if } t < e \\ bdt & \text{otherwise} \end{cases}$$

Using this weight function, we fulfill Equation 2.5. Furthermore, we make sure that the lower bounds stay consistent. A later arrival at a node never causes an earlier departure. Therefore, expanding the selection still only increases the length of any longest path. Note that the non-constant segment of $w(e_{c,n}, t)$ makes sure that the departure time stays constant during this period. It is $t + w(e_{c,n}, t) = t + e + bdt - t = e + bdt$. We make sure, that the course dwells as long as necessary to avoid the full extended dwell time. Figure 3.10 shows a plot of the weight function.

For $ert = (s, e, l, rt) \in ERT$, the modification of the weight function is analogue. The only difference is that we have to apply the modified weight function to multiple edges for each link operation. This is because multiple edges can have a weight modeling the runtime at the link (cf. Section 3.1.4). For each edge, we have to consider the activity-dependent base runtime in the modified weight function. However, the extended runtime stays the same independent of the activities.

3.1.9 Respecting the realized schedule

So far, the proposed model assumes that the behavior of courses before the snapshot time can be altered. However, we have to respect the realized schedule up to the snapshot time (cf. Equation 2.3). Therefore, we need to take additional measures to fix the start and completion time of any operation that takes place before the snapshot time. We also need to make sure that operations that are not explicitly listed in the realized schedule are not started before the snapshot time.

Let $si \in S_c$ be a schedule item. Let n and l be the node and link operations created for si. We say that n is realized if there exists $rsi \in RSI_c, rsi \neq \emptyset$ with

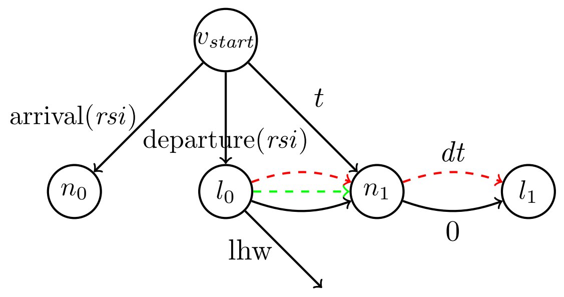


FIGURE 3.11: Transition from realized to non-realized operations

index(si) = index(rsi). l is realized if additionally departure(rsi) $\neq -1$. We do not connect two consecutive realized operations directly as we did in all prior sections. Rather, we create fixed edges from v_{start} to any realized operation. The weight of those edges is the start time of the operations. This is arrival(rsi) in the case of a node operation and departure(rsi) in the case of a link operation. Also, we add a fixed edge from v_{start} to the first non-realized operation of any rolling stock duty. Its weight is the snapshot time t. Figure 3.11 shows an example of this construction.

The realized operations seem redundant in the first place. However, we need them for modeling headway constraints between realized and non-realized operations. Assume that we are modeling a track headway constraint between a realized node operation n_0 and a non-realized operation n_1. They belong to the courses c_0 and c_1 respectively. Then, we set $T_{n_0} = \{\text{track}(rsi)\}$. Furthermore, we do not have to create the alternative pair p_{n_0,n_1}. This is because we know that the realized operation takes place before the non-realized one by definition. Therefore, we only create the direction of the constraint that assumes that c_0 enters the node first. Then, instead of using alternative edges, we use fixed edges for enforcing the constraint. Apart from that, we also use fixed edges for link headway constraints. For a realized link operation l_0 $HW_{l_0,in}$ becomes empty. $HW_{l_0,out}$ might be a singleton depending on whether the succeeding node operation is also realized.

Lastly, we connect the last realized operation to the first non-realized operation of every rolling stock duty. We do this the same way as we would connect two non-realized operations.

Removing redundant operations

Even when considering headway constraints, some of the realized operations may be redundant. This is because their fixed time is way before the snapshot time. Therefore, they cannot have any impact on non-realized operations. We calculate the maximum potential link headway

$$hw_{max} = \max_{\substack{l \in L \\ a_i \in Activities}} (\text{lhw}(l, a_0, a_1, a_2, a_3))$$

and the maximum potential runtime

$$rt_{max} = \max_{\substack{l \in L \\ a_i \in Activities}} (\text{mrt}(l, a_0, a_1))$$

We then prune any realized operation whose start time is before $t - \max(hw_{max}, rt_{max})$.

Furthermore, we also remove measure nodes m_c for courses that are fully realized. More formally, for courses c with $|S_c| = |RS_c|$. In those cases, $\mathrm{lp}_{\mathcal{S}}(m_c)$ is constant independent of the selection $\mathcal{S}$. Therefore, we do not need to calculate it dynamically.

3.1.10 Limits of the model

Our proposed approach does not apply to the full 2022 RAS problem. We restrict our view on the amendments re-timing, re-platforming and stop-skipping. Incorporating course cancellation and modification of rolling stock duties would add another degree of freedom. It is theoretically possible to express those amendments using an alternative graph. However, it would increase the size of the resulting graph even more. We already introduced a lot of additional vertices and edges compared to the graph for the CRFR problem. Therefore, it is more promising to handle those amendments differently. The basic idea is to solve multiple alternative graphs where each graph models another combination of cancellations and modified rolling stock duties. The process of finding the best amendments is then left to a higher-order search scheme. This is comparable to the approach of [Cor+10]. There Corman et al. solve the full conflict detection problem by applying a taboo search across multiple CRFR problems.

However, due to time constraints, we restrict the approach of this thesis to the limited set of amendments.

3.1.11 Extracting the amended timetable from a selection

We want to extract a solution to the 2022 RAS problem from every complete consistent selection. For the job-shop scheduling problem, this is possible by retrieving the start time of each operation o from $\mathrm{lp}_{\mathcal{S}}(o)$. With our model, we can use a similar approach. However, we have to consider some additional information for constructing the amended timetable.

Let $\mathcal{S}$ be a complete consistent selection. Also, let $\mathcal{T} = (C, \mathrm{cdir}, \mathrm{ccat}, (S_c)_{c\in C}, R)$ be the original timetable model. We can construct the amended timetable $\mathcal{T}' = (C', \mathrm{cdir}', \mathrm{ccat}', (S'_c)_{c\in C'}, R')$ from the selection $\mathcal{S}$. First, $C' = C, \mathrm{cdir}' = \mathrm{cdir}, \mathrm{ccat}' = \mathrm{ccat}$ and $R' = R$. This is because our model is not capable of adapting the rolling stock duties and introducing new courses. We only amend the schedules of existing courses (cf. Section 3.1.10). Now, let $si \in S_c$ be a planned schedule item. Also, let n and l be the node and link operation created for si. Finally, let $si' = (S'_c)_{\mathrm{index}(si)}$ be the corresponding schedule item in the amended schedule. Then, we can extract the arrival and departure times from the longest paths to the operations. Namely, $\mathrm{arrival}(si') = \mathrm{lp}_{\mathcal{S}}(n)$ and $\mathrm{departure}(si') = \mathrm{lp}_{\mathcal{S}}(l)$. We can also extract the chosen activity. Let $p = (a_{Pass}, a_{Stop})$ be the alternative pair created for deciding whether the course stops at $\mathrm{node}(si)$. Then

$$\mathrm{activity}(si') = \begin{cases} \mathit{Pass} & \text{if } a_{Pass} \subseteq \mathcal{S} \\ \mathit{Stop} & \text{if } a_{Stop} \subseteq \mathcal{S} \end{cases}$$

If no such pair exists, we use the activity that we predetermined when creating the operations. Apart from that, we can extract $\mathrm{track}(si')$ from the respective alternative pairs. To achieve this, we reverse the construction from Section 3.1.6.

We refer to the amended timetable created from the complete consistent selection $\mathcal{S}$ as $\mathcal{T}'(\mathcal{S})$.

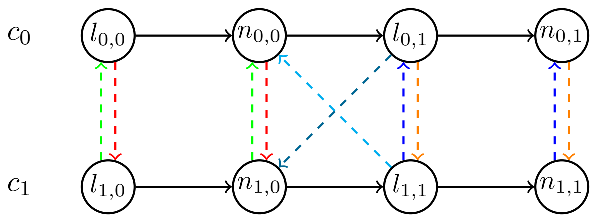


FIGURE 3.12: Simplified representation of two courses traveling along the same single-track nodes and links. Alternative pairs $p_{l_0} = (a_{l_0,0}, a_{l_0,1}) =$ (🟥, 🟩), $p_{n_0} = (a_{n_0,0}, a_{n_0,1}) =$ (🟦, 🟦) and $p_{l_1} = (a_{l_1,0}, a_{l_1,1}) =$ (🟧, 🟦).

3.2 Static implications

For an efficient computation of the branch-and-bound procedure, static implications are important. In this section, we adapt the static implications introduced in Section 2.3.3 to our modified model. For this adaption, we need to regard the fact that courses can overtake each other at nodes with multiple tracks. First, Section 3.2.1 considers static implications between courses traveling in the same direction. Then, Section 3.2.2 handles implications between courses traveling in different directions. Finally, Section 3.2.3 introduces a new type of static implications for headway constraints with courses of the same duty.

3.2.1 In the same direction

Static implications between courses traveling in the same direction were one result of Section 2.3.3. However, it relied on the strict definition of block sections and the fact that trains do not overtake each other within a block section. In the 2022 RAS problem, courses still cannot overtake each other within links. However, at nodes overtaking is a possibility. As a conclusion, our static implications will be weaker than the existing ones. In the following, we will first define an example scenario. Then, we will specify the static implications, given that there is only a single track available at the exemplary node. Last, we will explain which of those implications are invalid, given that there are multiple available tracks.

Let $c_0, c_1 \in C$ be two courses of different duties traveling in the same direction along the same nodes and links. Let l_0, n_0, l_1, n_1 be those nodes and links. Let $l_{i,j}$ and $n_{i,j}$ be the operations of course c_i at link l_j or node n_j respectively. Then, there are headway constraints between the two courses. Let $p_{l_0} = (a_{l_0,0}, a_{l_0,1}) \in A$ be the alternative pair modeling the headway constraint between c_0 and c_1 at link l_0. Here $a_{l_0,0}$ is the alternative modeling that c_0 arrives at the link first. Finally, let p_{n_0} and p_{l_1} be defined the same way as p_{l_0}.

We assume for now that the courses can only use the same single track at node n_0. More formally, $T_{n_{0,0}} = T_{n_{1,0}}$ and $|T_{n_{0,0}}| = 1$. Figure 3.12 depicts this scenario in a simplified way. It only contains a subset of the vertices and alternative edges. Nonetheless, we can tell from the figure that $a_{n_0,0} \leftrightsquigarrow a_{l_0,0}$ and $a_{n_0,0} \leftrightsquigarrow a_{l_1,0}$. Symmetrically, $a_{n_0,1} \leftrightsquigarrow a_{l_0,1}$ and $a_{n_0,1} \leftrightsquigarrow a_{l_1,1}$. We can check those implications by adding the non-implied alternative to the graph already containing the precedence of the implication. In this case, we get a cycle that makes our selection inconsistent. For example, the edges $(l_{0,1}, n_{1,0}) \in a_{n_0,0}$, $(n_{1,0}, n_{0,0}) \in a_{l_0,1}$ and $(n_{0,0}, l_{0,1}) \in F$ form a cycle. Formally, this reasoning corresponds to a modified version of Theorem 2.3.1 that is adjusted for our alternative graph definition. Descriptively, those implications

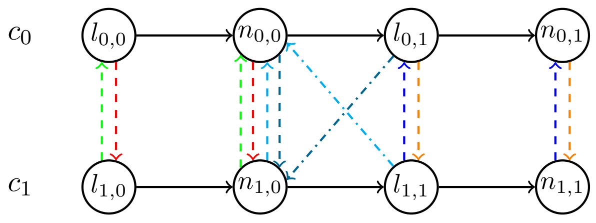


FIGURE 3.13: Simplified representation of two courses traveling along the same multi-track nodes and links. Alternative pairs $p_{l_0} = (a_{l_0,0}, a_{l_0,1}) = $ (🟥, 🟩), $p_{n_0} = (a_{n_0,0}, a_{n_0,1}) = $ (🟦, 🟦) and $p_{l_1} = (a_{l_1,0}, a_{l_1,1}) = $ (🟧, 🟦).

are still equivalent to the fact that two courses cannot overtake each other on a single track.

If the number of available tracks is higher than one, not all implications are valid anymore. Some of the edges of the alternative pair p_{n_0} are guarded by track decisions. This means that there is no direct edge $(l_{0,1}, n_{1,0})$ nor $(l_{1,1}, n_{0,0})$. Therefore, the cycles constructed above are no longer part of the graph. However, we have the additional alternative edges $(n_{0,0}, n_{1,0}) \in a_{n_0,0}$ and $(n_{1,0}, n_{0,0}) \in a_{n_0,1}$. Figure 3.13 shows this modified scenario. Note that in the figure "-·-·" does not depict a simple alternative edge. It depicts an edge that is potentially guarded by track decisions (cf. Section 3.1.6). All other depicted edges exist in our model.

We can also apply Theorem 2.3.1 in the multi-track scenario. We get the implications $a_{n_0,0} \leftrightsquigarrow a_{l_0,0}$ and $a_{n_0,1} \leftrightsquigarrow a_{l_0,1}$. We again comprehend these implications by looking at the potential cycles in the figure. Because of the guarded edges, we only have static implications between the alternative pair of the prior link headway and the track headway. But we do not have implications between the track headway and the following link headway. Descriptively, this is equivalent to the fact that courses may not overtake each other inside a link. So, a course that enters a link first will also enter the next node first. However, they may overtake each other at a node that has multiple available tracks. So, a course that enters a node first must not enter the next link first.

3.2.2 In different directions

The other kind of static implications introduced in Section 2.3.3 are static implications between courses traveling in different directions through the same block sections. In the 2022 RAS problem, links in different directions are almost always different resources. The few exceptions are links with direction *Both*. Apart from those, the same link cannot be used by courses traveling in different directions. However, courses traveling in different directions can interfere with each other at nodes n where $\text{tracks}(n, \textit{Eastbound}) \cap \text{tracks}(n, \textit{Westbound}) \neq \emptyset$. As the number of links with direction *Both* is neglectable, we only construct static implications between track headway constraints.

Let $c_0, c_1 \in C$ be two courses of different duties and with different directions that travel along the same nodes. Let $n_0, n_1 \in N$ be two of those nodes. c_0 visits n_0 first while c_1 visits n_1 first. The corresponding operations are $n_{0,0}, n_{0,1}$ for c_0 and $n_{1,0}, n_{1,1}$ for c_1. Also, let $T_{n_{0,0}} \cap T_{n_{1,0}} \neq \emptyset$ and $T_{n_{0,1}} \cap T_{n_{1,1}} \neq \emptyset$. Then, there are alternative pairs $p_{n_0} = (a_{n_0,0}, a_{n_0,1})$ and $p_{n_1} = (a_{n_1,0}, a_{n_1,1})$ modeling the track headway constraints. Here $a_{n_0,0}$ is the alternative modeling that c_0 arrives first at n_0. The other alternatives are defined symmetrically. For this discussion, we assume that there are multiple tracks available for each course at n_0 and n_1. If this is not the

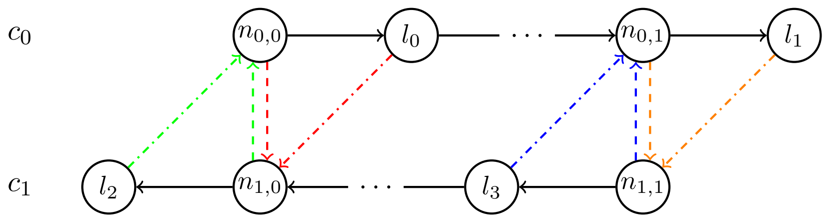


FIGURE 3.14: Simplified representation of two courses traveling along the same nodes and links in different directions with alternative pairs $p_{n_0} = (a_{n_0,0}, a_{n_0,1}) = (🟥, 🟩)$ and $p_{n_1} = (a_{n_1,0}, a_{n_1,1}) = (🟧, 🟦)$.

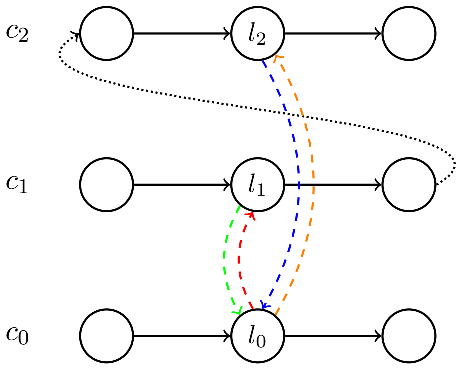


FIGURE 3.15: Simplified representation of three courses using the same link. c_1 and c_2 belong to the same duty. $p_{l_0,l_1} = (a_0, a_1) = (🟥, 🟩)$ and $p_{l_0,l_1} = (a_2, a_3) = (🟧, 🟦)$.

case one can make an analogue argument. Figure 3.14 shows a simplified subgraph depicting this scenario. Note that "-·-·" again represents guarded edges.

We can apply Theorem 2.3.1 once more to find static implications. We can see that $a_{n_1,0} \rightsquigarrow a_{n_0,0}$ and $a_{n_0,1} \rightsquigarrow a_{n_1,1}$. As in Section 2.3.3 the reverse implications do not hold. Again, one can comprehend those implications by finding potential cycles in the subgraph depicted by Figure 3.14. From the figure, one can also conclude, why the reverse implications do not hold. The implications express that c_0 cannot be at n_1 before c_1 if c_1 is at n_0 before c_0 and the symmetric case. The resulting cycles express the deadlock that would result from such a scenario.

3.2.3 For duty consistency

Our model allows for another kind of static implications that was not discussed in Section 2.3.3. It is applicable whenever a rolling stock duty uses the same resource multiple times. It models that the headways between a course from one duty and two or more courses of another duty must be consistent with the order of the courses in the other duty.

Let $\ell \in L$ be a link and $c_0, c_1, c_2 \in C$ three courses with $\text{duty}(c_0) = d_0 \in R$ and $\text{duty}(c_1) = \text{duty}(c_2) = d_1 \in R$. We assume that the duty element of c_2 has a higher index in d_1 than the duty element of c_1. All of the courses pass the link in the same direction $dir \in Direction$. The corresponding link operations are $l_0, \ldots, l_2$ for $c_0, \ldots, c_2$. Now, let $p_{l_0,l_1} = (a_0, a_1)$ and $p_{l_0,l_2} = (a_2, a_3)$ be the alternative pairs that we created for modeling the link headways. Figure 3.15 shows a simplified subgraph of this scenario. The dotted line depicts a sequence of fixed edges that connect the operations of the different courses. This sequence must exist as both c_1 and c_2 belong to the same duty.

We can construct the static implications again using Theorem 2.3.1. The first implication is $a_0 \rightsquigarrow a_2$. It expresses the fact that if c_0 uses ℓ before c_1 it must also use it before c_2. This must be the case, as c_1 is served by the same rolling stock as c_2 and before c_2. Similarly, we have $a_3 \rightsquigarrow a_1$. If c_0 uses ℓ after c_2 it must also use it after c_1. The potential cycles causing those implications can be seen in the figure.

By a similar construction, we can also create such duty consistency static implications for track headways. Then, the three courses c_0, c_1, c_2 must use the same node. Additionally, $T_{n_0} \cap T_{n_1} \neq \emptyset$ and $T_{n_0} \cap T_{n_2} \neq \emptyset$ must hold for the respective node operations $n_0, \dots, n_2$.

3.3 Adapting the branch-and-bound algorithm

Section 2.2.4 showed that we can use a branch-and-bound algorithm to find the best selection for a given alternative graph. However, the details of the algorithm depend on the specific problem. Those were, for example, the concrete search strategy and the selection of the next alternative pair to branch on. Also, the algorithm presented so far only considers job-shop scheduling problems. Our extended model introduced in Section 3.1 is more elaborate than the models for the job-shop scheduling problems. Furthermore, the objective function of the 2022 RAS problem does not correspond to the length of a single longest path. Therefore, the computation of lower bounds is more complicated. Altogether, we have to adapt the branch-and-bound procedure to work efficiently with our model. We present the different adaptions in the following sections. First, Section 3.3.1 shows how we can bound the objective function given a partial selection. Second, Section 3.3.2 defines a strategy for selecting the next alternative pair to branch on. Then, Section 3.3.3 introduces our strategy for exploring the search space. Lastly, Section 3.3.4 explains how we can find an initial selection that can be used as the current best solution right from the start of the algorithm.

3.3.1 Computing lower bounds

The computation of lower bounds is important for pruning subtrees in a branch-and-bound algorithm. When solving the job-shop scheduling problem, we can extract a lower bound for the makespan by calculating the longest path from the start to the end vertex. Contrarily, the 2022 RAS problem, has a more elaborate objective function. The total penalty P_{total} is the sum of the three components P_{ss}, P_{dd} and P_f (cf. Section 2.4.5). In the following, we describe the computation of the lower bounds for each component individually. For each of those components P_x let $P_x^{\mathcal{S}}$ be the penalty of $\mathcal{T}'(\mathcal{S})$. We want to find a function $\mathrm{LB}_x(\mathcal{S})$ such that $\mathrm{LB}_x(\mathcal{S}) \leq P_x^{\mathcal{S}'}$ for all extensions $\mathcal{S}'$ of $\mathcal{S}$.

Destination Delay Penalty

The first component is the destination delay penalty P_{dd}. Let $\mathcal{S}$ be a complete consistent selection. To compute $P_{ss}^{\mathcal{S}}$, we need to know the destination delay dd_c for all courses with $\mathrm{ccat}(c) = \mathit{Passenger}$. For a course c that is not fully realized, this is $\mathrm{lp}_{\mathcal{S}}(m_c)$ by construction. For fully realized courses, there does not exist a measure node. However, for those the destination delay is constant independent of the selection. Let CP_{dd} be this constant part. Then

$$\begin{aligned}
P_{dd}^{\mathcal{S}} &= \sum_{m_c \in M} P_{dd,c} + CP_{dd} \\
&= \sum_{\substack{m_c \in M \\ \mathrm{lp}_{\mathcal{S}}(m_c) \geq 180}} \mathrm{lp}_{\mathcal{S}}(m_c) \cdot \frac{125}{60} + CP_{dd}
\end{aligned}$$

This function is monotonically increasing in all $\mathrm{lp}_{\mathcal{S}}(m_c)$. Also, for an arbitrary selection $\mathcal{S}'$, $\mathrm{lp}'_{\mathcal{S}}(m_c) \leq \mathrm{lp}_{\mathcal{S}}(m_c)$ for any extension $\mathcal{S}$ of $\mathcal{S}'$. Therefore, we can define

$$\mathrm{LB}_{dd}(\mathcal{S}') = \sum_{\substack{m_c \in M \\ \mathrm{lp}'_{\mathcal{S}}(m_c) \geq 180}} \mathrm{lp}'_{\mathcal{S}}(m_c) \cdot \frac{125}{60} + CP_{dd}$$

Then, $\mathrm{LB}_{dd}(\mathcal{S}') \leq P_{dd}^{\mathcal{S}}$. For complete consistent selections, equality holds.

Skipped stop penalty

The skipped stop penalty P_{ss} cannot be calculated from the length of longest paths. We rather have to consider which alternatives are part of the selection. Furthermore, $P_{ss,c}$ does not necessarily increase with an increased number of skipped stops. This is because of the counter-intuitive definition of ssf_c (cf. Equation 2.1). For instance, let $si_0, \ldots, si_2 \in S_c$ be three schedule items with $\mathrm{activity}(si_i) = \mathit{Stop}$. Let $\mathrm{bsv}(s_i) = 10 \cdot (i+1)$. First, assume that the course only skips the stop at si_2. Then

$$P_{ss,c} = \mathrm{bsv}(si_2)\, \mathrm{ssf}_c(si_2) = 30 \cdot 35 = 1050$$

Now, assume that the course skips all three planned stops. Then

$$\begin{aligned}
P_{ss,c} &= \mathrm{bsv}(si_0)\, \mathrm{ssf}_c(si_0) + \mathrm{bsv}(si_1)\, \mathrm{ssf}_c(si_1) + \mathrm{bsv}(si_2)\, \mathrm{ssf}_c(si_2) \\
&= 10 \cdot 35 + 20 \cdot 15 + 30 \cdot 1 \\
&= 680
\end{aligned}$$

As a consequence, for a partial selection $\mathcal{S}'$ we cannot compute $\mathrm{LB}_{ss}(\mathcal{S}')$ by only considering the stop/pass alternative pairs that are already selected in $\mathcal{S}'$. We also need to consider the remaining stop/pass alternative pairs not in $\mathcal{S}'$.

Let $c \in C$ be a course. Then, let $S_{c,\mathit{Stop}} \subseteq S_c$ be the subsequence of the schedule containing all schedule items with activity *Stop*. Furthermore, let $SS_c^{\mathcal{S}'}$ be the set of schedule items from $S_{c,\mathit{Stop}}$ whose stop is already skipped based on $\mathcal{S}'$. More formally, for each $si \in SS_c^{\mathcal{S}'}$ there is a stop/pass alternative pair $p = (a_{\mathit{Pass}}, a_{\mathit{Stop}})$ created for si with $a_{\mathit{Pass}} \subseteq \mathcal{S}'$. Finally, let $PSS_c^{\mathcal{S}'}$ be the set of schedule items from $S_{c,\mathit{Stop}}$ whose activity is not defined based on $\mathcal{S}'$. In other words, the set of **p**otentially **s**kipped **s**tops. Formally, for each $si \in PSS_c^{\mathcal{S}'}$, there is a stop/pass alternative pair $p = (a_{\mathit{Pass}}, a_{\mathit{Stop}})$ created for si with neither $a_{\mathit{Pass}} \subseteq \mathcal{S}'$ nor $a_{\mathit{Stop}} \subseteq \mathcal{S}'$. $\hat{SS}_c^{\mathcal{S}'}$ and $\hat{PSS}_c^{\mathcal{S}'}$ are the sequences containing all elements of the respective sets sorted in ascending order based on $\mathrm{bsv}(\cdot)$.

Based on those definitions, we can specify $\mathrm{LB}_{ss,c}(\mathcal{S}')$ to correctly lower bound the skipped stop penalty for an individual course. It is defined to be the minimum of the following four sums.

$$\mathrm{bsv}((\hat{PSS}_c^{\mathcal{S}'})_0) \cdot 35 \quad + \mathrm{bsv}((\hat{PSS}_c^{\mathcal{S}'})_1) \cdot 15 \quad + \sum_{i=0}^{|SS_c^{\mathcal{S}'}|-1} \mathrm{bsv}((\hat{SS}_c^{\mathcal{S}'})_i) \tag{3.1}$$

$$\mathrm{bsv}((\hat{PSS}_c^{\mathcal{S}'})_0) \cdot 35 \quad + \mathrm{bsv}((\hat{SS}_c^{\mathcal{S}'})_0) \cdot 15 \quad + \sum_{i=1}^{|SS_c^{\mathcal{S}'}|-1} \mathrm{bsv}((\hat{SS}_c^{\mathcal{S}'})_i) \tag{3.2}$$

$$\mathrm{bsv}((\hat{SS}_c^{\mathcal{S}'})_0) \cdot 35 \quad + \mathrm{bsv}((\hat{PSS}_c^{\mathcal{S}'})_0) \cdot 15 \quad + \sum_{i=1}^{|SS_c^{\mathcal{S}'}|-1} \mathrm{bsv}((\hat{SS}_c^{\mathcal{S}'})_i) \tag{3.3}$$

$$\mathrm{bsv}((\hat{SS}_c^{\mathcal{S}'})_0) \cdot 35 \quad + \mathrm{bsv}((\hat{SS}_c^{\mathcal{S}'})_1) \cdot 15 \quad + \sum_{i=2}^{|SS_c^{\mathcal{S}'}|-1} \mathrm{bsv}((\hat{SS}_c^{\mathcal{S}'})_i) \tag{3.4}$$

Sum 3.1 models the case that the course additionally skips the two stops whose schedule items have the lowest base station value among those in PSS. Sum 3.2 and 3.3 both model the scenario that the course skips the one additional stop whose schedule item has the lowest base station value. Lastly, Sum 3.4 represents the scenario that no additional stop is skipped. Depending on the concrete base station values, any of those four alternatives might be the smallest. Therefore, the lower bound $\mathrm{LB}_{ss,c(\mathcal{S}')}$ is the minimum of all of them. All other scenarios cause a higher skipped stop penalty independent of the concrete base station values. Therefore, the computed lower bound is valid.

The sums are not well defined if the sequences have less than two elements. To remedy this, we say that the base station value of a non-existing element of $\hat{SS}_c^{\mathcal{S}'}$ evaluates to 0. On the other hand, the base station value of a non-existing element of $\hat{PSS}_c^{\mathcal{S}'}$ evaluates to ∞. This is consistent, as the sums that rely on skipping additional stops become infeasible if there are no additional stops that can be skipped. Furthermore, it nullifies the summands of Sum 3.4 if there are fewer than two skipped stops at all. Consequently, there is no penalty if no stops are skipped.

Frequency penalty

Currently, we do not compute lower bounds for the frequency penalty. Therefore, we set $\mathrm{LB}_f(\mathcal{S}) = 0$ for all selections $\mathcal{S}$. Furthermore, optimizing a criterion without lower bounds for it is not promising. Consequently, we ignore the frequency penalty when searching for an optimal selection. Our current approach only optimized for the destination delay and skipped stop penalty.

3.3.2 Choice selection

In the branch-and-bound algorithm, the order in which we consider the different choices impacts the overall performance of the algorithm heavily. With a good order, the algorithm may be able to prune entire subtrees quite early. In our concrete case, we have to decide which alternative pair to consider next given a partial selection $\mathcal{S}$. When solving the CRFR problem, in [DPP07] D'Ariano et al. always choose the unselected alternative pair with the alternative edge $e = (i, j)$ that maximizes

$$\mathrm{lp}_{\mathcal{S}}(v_{start}, i) + w(e) + \mathrm{lp}_{\mathcal{S}}(v_{j,end})$$

Their goal is to consider alternative pairs first that have a high impact on the objective function. The underlying idea is that the lower bounds for selections containing those alternative edges are high. As a consequence, they can be pruned early into the search.

However, this does not apply to our model. Our model does not contain a single end vertex v_{end} but rather multiple measure vertices m_c. Furthermore, our objective function is more complicated than the makespan. It additionally considers skipped stops. Therefore, we adapt the strategy for choosing the next alternative pair to branch on. Nonetheless, we try to adhere to the underlying idea.

Let $p = (a_0, a_1) \in A$ be an unselected alternative pair. Based on the current selection, we assign a score $\mathrm{sc}(a_i, \mathcal{S})$ to each of its alternatives. We define the score of the whole alternative pair to be $\mathrm{sc}(p, \mathcal{S}) = \max(\mathrm{sc}(a_0, \mathcal{S}), \mathrm{sc}(a_1, \mathcal{S}))$. Then, we branch on the alternative pair with the highest score. The score of the individual alternatives is the weighted sum of multiple components.

$$\mathrm{sc}(a_i, \mathcal{S}) = \alpha \cdot \mathrm{sc}_{dd}(a_i, \mathcal{S}) + \beta \cdot \mathrm{sc}_{ss}(a_i, \mathcal{S}) + \gamma \cdot \mathrm{sc}_g(a_i, \mathcal{S})$$

We explain the idea and definition of the different components in the following sections. α, β and γ are configurable parameters.

Destination delay

As in [DPP07], we want to emphasize branching on alternative pairs with edges that cause long paths. We want to capture this by $\mathrm{sc}_{dd}(a_i, \mathcal{S})$. To achieve this, we first need to define some utility functions. Let $e = (h, k) \in a_i$, then

$$d(e, c, \mathcal{S}) = \mathrm{lp}_{\mathcal{S}}(h) + w(e, \mathrm{lp}_{\mathcal{S}}(h)) + \mathrm{lp}_{\mathcal{S}}(k, m_c) - \mathrm{lp}_{\emptyset}(m_c)$$

$d(e, c, \mathcal{S})$ describes the impact an edge e has on $\mathrm{lp}_{\mathcal{S}}(m_c)$. It is defined to be the length of the longest path from v_{start} to m_c in $G(\mathcal{S} \cup \{e\})$ that uses the edge e. Additionally, we subtract the unavoidable primary delay of the course c to normalize $d(e, c, \mathcal{S})$ across multiple courses. The unavoidable primary delay is given by the length of the longest path to m_c in $G(\emptyset)$. We further define

$$d(e, \mathcal{S}) = \max_{c \in C}(d(e, c, \mathcal{S}))$$

It is the maximum impact an edge has across all courses. Finally the destination delay score of an alternative a_i under the selection *sel* is

$$\mathrm{sc}_{dd}(a_i, \mathcal{S}) = \max_{e \in a_i}(d(e, \mathcal{S}))$$

It is defined to be the maximum impact of all edges of the alternative. We note for later that $\mathrm{sc}_{dd}(a_i, \mathcal{S}) \leq \mathrm{sc}_{dd}(a_i, \mathcal{S}')$ for $\mathcal{S} \subseteq \mathcal{S}'$.

Stop pass choices

We also want to incentivize the consideration of stop/pass alternative pairs with a high potential impact on the skipped stop penalty P_{ss}. Let $p = (a_{pass}, a_{stop})$ be a stop/pass alternative pair. Let $c \in C$ and $si \in S_c$ be the course and schedule item for which p was created. If $\mathrm{ccat}(c) = \mathit{Passenger}$ and $\mathrm{activity}(si) = \mathit{Stop}$, we set

$$\mathrm{sc}_{ss}(a_{pass}, \mathcal{S}) = \mathrm{bsv}(si)$$

independent of the selection $\mathcal{S}$. In all other cases, we set $\mathrm{sc}_{ss}(a_i, \mathcal{S}) = 0$. This emphasizes pass alternatives at stations with a high base station value. As skipping stops at those stations has a high impact on the skipped stop penalty, it is aligned with our overall strategy.

Choice grouping

At last, we want to incentivize the consideration of choices, that are related to already selected alternative pairs. This is helpful if the selection of a single alternative does not have an impact on the objective function on its own. For example, consider choosing a track for a course at a node with more than two available tracks. Then, this decision is encoded in more than one alternative pair (cf. Section 3.1.6). Considering one of those alternative pairs individually does not have an impact on the objective function. Therefore, we also want to select the other track alternative pairs whenever one track alternative pair is selected. We formalize this concept by defining the function

$$\mathrm{rel} : A \times A \to \mathbb{B}$$

with the semantics

$$\mathrm{rel}(p, q) = \begin{cases} \mathit{true} & \text{if } p \text{ is related to } q \\ \mathit{false} & \text{otherwise} \end{cases}$$

Now, let $p, q \in A$ be two alternative pairs that were created to encode the used track of a course c at a node n. Then, we set $\mathrm{rel}(p, q) = \mathrm{rel}(q, p) = \mathit{true}$.

With these definitions, we can define our last score function. Let $p = (a_0, a_1) \in A$. Then, we define

$$\mathrm{sc}_g(a_i, \mathcal{S}) = \begin{cases} 1 & \text{if } \exists q = (a_2, b_3) \in A : (a_2 \subseteq \mathcal{S} \vee a_3 \subseteq \mathcal{S}) \wedge \mathrm{rel}(q, p) \\ 0 & \text{otherwise} \end{cases}$$

By setting γ sufficiently high, we can ensure that related alternative pairs are considered immediately. Also, we again note for later that $\mathrm{sc}_g(a_i, \mathcal{S}) \leq \mathrm{sc}_g(a_i, \mathcal{S}')$ for $\mathcal{S} \subseteq \mathcal{S}'$.

3.3.3 Search strategy

The concrete search strategy of our branch-and-bound algorithm is a mixture of best-first and depth-first search. It is configurable by a parameter χ. Whenever we branch on a selection, we check whether we can immediately prune any of its children. Then, we add every non-pruned child selection to a stack. Now, whenever we decide which state we want to branch on next, we look at the top χ selections on the stack. We remove the selection $\mathcal{S}$ from the stack, that has the lowest lower bound value among those χ selections. Then, we branch on this selection. The search space is fully explored once the stack is empty. Additionally, whenever we find a new best selection, we eagerly remove any selections from the stack that can be pruned. Figure 3.16 shows an example of one search step with the described search strategy. For $\chi = 1$ the strategy corresponds to depth-first search. For $\chi = \infty$ it is equal to best-first search.

3.3.4 Finding an initial complete selection

For speeding up the branch-and-bound algorithm, it is helpful to find an initial complete selection $\mathcal{S}_{ini}$. This selection can be stored as the best solution found so far at

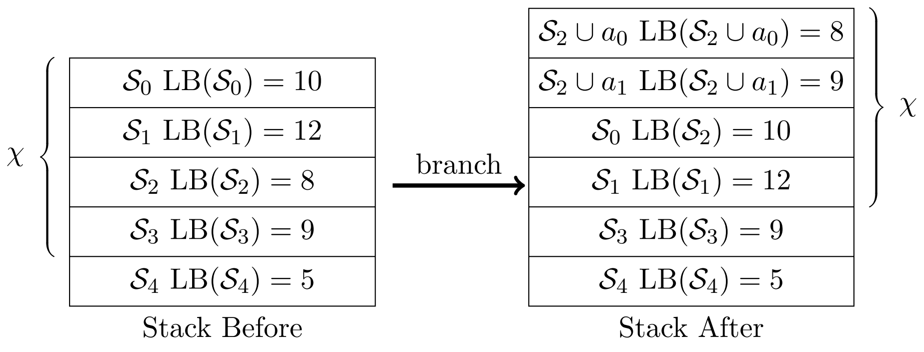

$\mathcal{S}_0$ LB($\mathcal{S}_0$) = 10
$\mathcal{S}_1$ LB($\mathcal{S}_1$) = 12
$\mathcal{S}_2$ LB($\mathcal{S}_2$) = 8
$\mathcal{S}_3$ LB($\mathcal{S}_3$) = 9
$\mathcal{S}_4$ LB($\mathcal{S}_4$) = 5
χ
branch
$\mathcal{S}_2 \cup a_0$ LB($\mathcal{S}_2 \cup a_0$) = 8
$\mathcal{S}_2 \cup a_1$ LB($\mathcal{S}_2 \cup a_1$) = 9
$\mathcal{S}_0$ LB($\mathcal{S}_2$) = 10
$\mathcal{S}_1$ LB($\mathcal{S}_1$) = 12
$\mathcal{S}_3$ LB($\mathcal{S}_3$) = 9
$\mathcal{S}_4$ LB($\mathcal{S}_4$) = 5
χ
Stack Before
Stack After

FIGURE 3.16: The process of branching on the selection with the lowest lower bounds among the top χ selections on the stack.

the beginning of the algorithm. As a consequence, we can prune selections with a worse lower bound right from the start. We use a first-come, first-served approach for finding an initial selection.

We first consider all alternative pairs that are not related to the order of trains. Those are stop/pass and track alternative pairs. We add the alternative to the selection that represents the decision of the original timetable. For instance, let $si \in S_c$ be a schedule item with activity$(si) = Stop$. Also, let $p = (a_{Pass}, a_{Stop})$ be the stop/pass alternative pair created for this schedule item. Then, $a_{Stop} \in \mathcal{S}_{ini}$.

Next, we consider link and track headway alternative pairs. Therefore, we go through the operations of each duty. In each step, we choose the operation o with the lowest $\text{lp}_{\mathcal{S}_{ini}}(o)$ that we have not yet considered. Let c be the course corresponding to operation o. There are two cases. First, let o be a node operation. Then, we do the following for all track headway alternative pairs that are not already selected in $\mathcal{S}_{ini}$. We add the alternative that models that c arrives at the node first to $\mathcal{S}_{ini}$. In the other case, o is a link operation. Then, we do the following for all link headway alternative pairs that are not already selected in $\mathcal{S}_{ini}$. We add the alternative to $\mathcal{S}_{ini}$ that models that c arrives at the link first. We are finished after all operations have been chosen once or if $\mathcal{S}_{ini}$ is complete.

3.4 Updating state metadata

For computing lower bounds and for selecting the next alternative pair to branch on, there are some functions that need to be computed for every selection $\mathcal{S}$. Those are, for example, the length of the longest path to some node $\text{lp}_{\mathcal{S}}(n)$ or the score of an alternative pair $\text{sc}(p, \mathcal{S})$. Computing those values from scratch for every selection $\mathcal{S}$ can be time-consuming. However, when branching on a selection $\mathcal{S}$, we can compute the functions for the child selection $\mathcal{S}'$ based on their values for the parent selection. If this is faster than the computation from scratch, we speed up the entire algorithm.

From an implementation perspective, we store this metadata along the selection on the stack. We compute the metadata for the child selections whenever we branch on a selection. The updated metadata is then again added to the stack along the two child selections.

In the following sections, we describe the computation and updating of different parts of the metadata. We start by maintaining the length of longest path $\text{lp}_{\mathcal{S}}(\cdot)$ in Section 3.4.1. Then, we continue with updating the values necessary for computing $\text{LB}_{ss}(\mathcal{S})$ in Section 3.4.2. We finish with the data necessary for the choice selection in Section 3.4.3. After that, we introduce two techniques that further speed up the computation and optimize the memory usage of our algorithm in Sections 3.4.4 and 3.4.5.

3.4.1 Topological order & longest paths

In this section, we describe how we can initialize and update the longest paths $\mathrm{lp}_{\mathcal{S}}(v_{start}, \cdot)$ efficiently. Notice that we constructed the alternative graph G in such a way that $G(\mathcal{S})$ is a directed acyclic graph for every consistent selection $\mathcal{S}$. Therefore, we can compute a topological order on $G(\mathcal{S})$. For a vertex $v \in V$ we denote its computed topological order in $G(\mathcal{S})$ as $\mathrm{top}_{\mathcal{S}}(v)$. We compute the initial topological order using a depth-first search. It is the reverse postorder of the DFS (cf. [Cor+22, Chapter 20.4]). With this topological order, we can compute $\mathrm{lp}(\cdot)$ in linear time (cf. [Cor+22, Chapter 22.2]). The concrete computation is shown as part of Algorithm 1. We go through all vertices in order and update the path lengths of all of their successors if necessary.

Let $\mathcal{S}'$ be a child selection of $\mathcal{S}$ and $\mathrm{top}_{\mathcal{S}}(\cdot)$ a topological order of $G(\mathcal{S})$. Then, we do not need to compute a topological order of $G(\mathcal{S}')$ from scratch. Rather, we can update $\mathrm{top}_{\mathcal{S}}(\cdot)$ to $\mathrm{top}_{\mathcal{S}'}(\cdot)$ as $\mathcal{S} \subseteq \mathcal{S}'$. To achieve this, we use the algorithm presented in [PK10]. It can maintain a topological order of a directed acyclic graph when batch inserting new edges. It is applicable in our case as $G(\mathcal{S}')$ only differs from $G(S)$ by the edges $\mathcal{S}' \setminus \mathcal{S}$. With the new topological order of $G(\mathcal{S}')$, we can also calculate $\mathrm{lp}_{\mathcal{S}'}(\cdot)$ based on $\mathrm{lp}_{\mathcal{S}}(\cdot)$. As shown in Algorithm 1, we start by setting $\mathrm{lp}_{\mathcal{S}'}(\cdot) = \mathrm{lp}_{\mathcal{S}}(\cdot)$. Then, we only update $\mathrm{lp}_{\mathcal{S}'}(v)$ for a vertex v if it is impacted by the edges in $\mathcal{S}' \setminus \mathcal{S}$ (cf. line 14ff). This way, we avoid the consideration of vertices v for which $\mathrm{lp}_{\mathcal{S}}(v)$ does not change. After that, we go through the remaining vertices in topological order and forward the updated path length to their successors if necessary. In a concrete implementation, we can speed up the loop in line 20. Therefore, we keep track of the number of vertices in the *needsUpdate* set. We can terminate the loop once the set is empty. Thereby, we avoid iteratively checking the condition in line 21. Furthermore, in the loop, we can ignore all vertices with a smaller topological order than w. Here w is the vertex with the smallest topological order in *needsUpdate* after the completion of the first loop in line 14. With this adaption, we make sure that we skip the initial segment of the graph that is not affected by the new edges.

3.4.2 Skipped stops

This section handles the updating of the values necessary to compute $\mathrm{LB}_{ss}(\mathcal{S})$. We do not compute $\mathrm{LB}_{ss}(\mathcal{S})$ from scratch for every selection $\mathcal{S}$. We rather store some updatable metadata from which it can be computed efficiently. For every course $c \in C$ with $\mathrm{ccat}(c) = \mathit{Passenger}$ this metadata is

$$\begin{aligned}
\mathit{smlst}_{c,\mathcal{S}} &:= \mathrm{bsv}((\hat{SS}_c^{\mathcal{S}})_0) \\
\mathit{scSmlst}_{c,\mathcal{S}} &:= \mathrm{bsv}((\hat{SS}_c^{\mathcal{S}'})_1) \\
\mathit{remainingSum}_{c,\mathcal{S}} &:= \sum_{i=2}^{|SS_c^{\mathcal{S}}|-1} \mathrm{bsv}((\hat{SS}_c^{\mathcal{S}})_i) \\
\mathit{potBSV}_{c,\mathcal{S}} &:= (\mathrm{bsv}((\hat{PSS}_c^{\mathcal{S}})_0), \dots, \mathrm{bsv}((\hat{PSS}_c^{\mathcal{S}})_{|PSS_c^{\mathcal{S}}|-1}))
\end{aligned}$$

As discussed in Section 3.3.1, this metadata suffices for computing $\mathrm{LB}_{ss,c}(\mathcal{S})$. Furthermore, given a selection $\mathcal{S}' = \mathcal{S} \cup a_i$ for $p = (a_0, a_1) \in A$ we can update the

Algorithm 1 Algorithm for computing and updating $\mathrm{lp}_{\mathcal{S}}$

```
1:  procedure InitialPaths($\mathcal{S}$)
2:      needsUpdate = $\{v_{start}\}$
3:      $\mathrm{lp}_{\mathcal{S}}(\cdot) \leftarrow -\infty$
4:      $\mathrm{lp}_{\mathcal{S}}(v_{start}) \leftarrow 0$
5:      for $v \in V$ in order of $\mathrm{top}_{\mathcal{S}}(\cdot)$ do
6:          if $v \in$ needsUpdate then
7:              updateVertex($v, \mathcal{S}$)
8:          end if
9:      end for
10: end procedure

11: procedure UpdatePaths($\mathcal{S}, \mathcal{S}'$)
12:     needsUpdate = $\{\}$
13:     $\mathrm{lp}_{\mathcal{S}'}(\cdot) \leftarrow \mathrm{lp}_{\mathcal{S}}(\cdot)$
14:     for $(v, w) \in \mathcal{S}' \setminus \mathcal{S}$ do
15:         relaxed = relaxEdge($(v, w)$)
16:         if relaxed then
17:             needsUpdate $\leftarrow$ needsUpdate $\cup \{w\}$
18:         end if
19:     end for
20:     for $v \in V$ in order of $\mathrm{top}_{\mathcal{S}'}(\cdot)$ do
21:         if $v \in$ needsUpdate then
22:             updateVertex($v, \mathcal{S}'$)
23:         end if
24:     end for
25: end procedure

26: procedure updateVertex($v, \mathcal{S}$)
27:     if $\mathrm{lp}_{\mathcal{S}}(v) = -\infty$ then
28:         needsUpdate = needsUpdate $\setminus \{v\}$
29:         return 0
30:     end if
31:     for $(v, w) \in F \cup \mathcal{S}$ do
32:         relaxed $\leftarrow$ RelaxEdge($(v, w)$)
33:         if relaxed $\wedge\ w \notin$ needsUpdate then
34:             needsUpdate $\leftarrow$ needsUpdate $\cup \{w\}$
35:         end if
36:     end for
37:     needsUpdate $\leftarrow$ needsUpdate $\setminus \{v\}$
38: end procedure

39: procedure relaxEdge($(v, w), \mathcal{S}$)
40:     if $\mathrm{lp}_{\mathcal{S}}(v) + w((v, w), \mathrm{lp}_{\mathcal{S}}(v)) > \mathrm{lp}_{\mathcal{S}}(w)$ then
41:         $\mathrm{lp}_{\mathcal{S}}(w) = \mathrm{lp}_{\mathcal{S}}(v) + w((v, w), \mathrm{lp}_{\mathcal{S}}(v))$
42:     end if
43: end procedure
```

metadata easily. If p is not a stop/pass alternative pair, nothing changes. As a consequence, the metadata of the child selection is identical to the parent's metadata. To discuss the more interesting case, let $p = (a_{Pass}, a_{Stop})$ be a stop/pass alternative pair of course c. Let $si \in S_c$ be the corresponding schedule item. In any case,

$$potBSV_{c,\mathcal{S}'} = potBSV_{c,\mathcal{S}} \setminus \mathrm{bsv}(si)$$

This is because the stop/pass alternative pair of si is now part of the selection. Therefore, it does no longer resemble a potentially skipped stop. If $a_i = a_{Pass}$, we also update the other values.

$$smlst_{c,\mathcal{S}'} = \begin{cases} smlst_{c,\mathcal{S}} & \text{if } \mathrm{bsv}(si) \geq smlst_{c,\mathcal{S}} \\ \mathrm{bsv}(si) & \text{if } \mathrm{bsv}(si) < smlst_{c,\mathcal{S}} \end{cases}$$

$$scSmlst_{c,\mathcal{S}'} = \begin{cases} scSmlst_{c,\mathcal{S}} & \text{if } \mathrm{bsv}(si) \geq scSmlst_{c,\mathcal{S}} \\ smlst_{c,\mathcal{S}} & \text{if } \mathrm{bsv}(si) < scSmlst_{c,\mathcal{S}} \wedge \mathrm{bsv}(si) < smlst_{c,\mathcal{S}} \\ \mathrm{bsv}(si) & \text{if } \mathrm{bsv}(si) < scSmlst_{c,\mathcal{S}} \wedge \mathrm{bsv}(si) \geq smlst_{c,\mathcal{S}} \end{cases}$$

$$remainingSum_{c,\mathcal{S}'} = \begin{cases} remainingSum_{c,\mathcal{S}} + \mathrm{bsv}(si) & \text{if } \mathrm{bsv}(si) \geq scSmlst_{c,\mathcal{S}} \\ remainingSum_{c,\mathcal{S}'} + scSmlst_{c,\mathcal{S}} & \text{if } \mathrm{bsv}(si) < scSmlst_{c,\mathcal{S}} \end{cases}$$

The updated values are consistent with their initial definition. The update mimics the inclusion of si in $\hat{SS}_c^{\mathcal{S}}$ to get $\hat{SS}_c^{\mathcal{S}'}$

3.4.3 Choice selection

Another function that we need to compute constantly during the branch-and-bound algorithm is the score of an unselected alternative pair $\mathrm{sc}(p, \mathcal{S})$. More precisely, we need to retrieve the unselected alternative pair with the highest overall score. To achieve this, we associate a priority queue containing all unselected alternative pairs to each selection on the stack. We refer to this queue as $Q_{\mathrm{sc}}^{\mathcal{S}}$. The order of the queue is determined by the score function of the alternative pairs. We initialize it by explicitly computing the score function for every alternative pair before adding it to the queue. Now, let $\mathcal{S}'$ be a child selection of $\mathcal{S}$. In this section, we explain the computation of $Q_{\mathrm{sc}}^{\mathcal{S}'}$ from $Q_{\mathrm{sc}}^{\mathcal{S}}$.

The first trivial modification necessary to obtain $Q_{\mathrm{sc}}^{\mathcal{S}'}$ is removing any alternative pairs from $Q_{\mathrm{sc}}^{\mathcal{S}}$ that are selected in $\mathcal{S}'$ but not in $\mathcal{S}'$. In the branch and bound algorithms those are the alternative pair considered at the last branch together with all alternative pairs of its static implications.

In the second step, we need to adapt the order of the remaining pairs. Therefore, we need to identify the set of unselected alternative pairs p for which $\mathrm{sc}(p, \mathcal{S}) \neq \mathrm{sc}(p, \mathcal{S}')$. We name this set $P_{\Delta\,\mathrm{sc}}$. The score function of an alternative pair $\mathrm{sc}(p, \mathcal{S}')$ is the maximum of the score functions of its alternatives. Those are a weighted sum of three components (cf. Section 3.3.2). Namely,

$$\mathrm{sc}(a_i, \mathcal{S}) = \alpha \cdot \mathrm{sc}_{dd}(a_i, \mathcal{S}) + \beta \cdot \mathrm{sc}_{ss}(a_i, \mathcal{S}) + \gamma \cdot \mathrm{sc}_{g}(a_i, \mathcal{S})$$

As the weights α, β and γ are constant, it may only change if one of the summands changes. In Section 3.3.2, we saw that $\mathrm{sc}_{ss}(a_i, \mathcal{S})$ is also constant with respect to the selection $\mathcal{S}$. Therefore, $\mathrm{sc}(a_i, \mathcal{S})$ may only change if $\mathrm{sc}_{dd}(a_i, \mathcal{S})$ or $\mathrm{sc}_{g}(a_i, \mathcal{S})$ changes. Furthermore, both $\mathrm{sc}_{dd}(a_i, \mathcal{S})$ and $\mathrm{sc}_{g}(a_i, \mathcal{S})$ can only increase when new alternative pairs are selected. As a consequence, the following equivalence holds:

$$\begin{aligned} & \mathrm{sc}(a_i, \mathcal{S}) \neq \mathrm{sc}(a_i, \mathcal{S}') \\ \iff & \mathrm{sc}(a_i, \mathcal{S}) < \mathrm{sc}(a_i, \mathcal{S}') \\ \iff & \mathrm{sc}_{dd}(a_i, \mathcal{S}) < \mathrm{sc}_{dd}(a_i, \mathcal{S}') \vee \mathrm{sc}_g(a_i, \mathcal{S}) < \mathrm{sc}_g(a_i, \mathcal{S}') \end{aligned} \tag{3.5}$$

For an alternative pair $p = (a_0, a_1)$, $\mathrm{sc}_g(a_0, \mathcal{S}) = \mathrm{sc}_g(a_1, \mathcal{S})$. Therefore, we can define $\mathrm{sc}_g(p, \mathcal{S}) = \mathrm{sc}_g(a_0, \mathcal{S})$. Then, we can compute the set of unselected alternative pairs for which $\mathrm{sc}_g(p, \mathcal{S}) < \mathrm{sc}_g(p, \mathcal{S}')$ quite easily. We name it $P_{\Delta\,\mathrm{sc}_g}$. Let P_{new} be the set of alternative pairs that are selected in $\mathcal{S}'$ but not in $\mathcal{S}$. Furthermore, let P_{old} be the set of alternative pairs that are selected in $\mathcal{S}$. Finally, let P_{rem} be the set of alternative pairs that are still unselected in $\mathcal{S}'$. Then

$$P_{\Delta\,\mathrm{sc}_g} = \{p \in P_{rem} \mid \exists q \in P_{new} : \mathrm{rel}(q, p) \wedge \nexists \hat{q} \in P_{old} : \mathrm{rel}(\hat{q}, p)\}$$

We know that $P_{\Delta\,\mathrm{sc}_g} \subseteq P_{\Delta\,\mathrm{sc}}$. This is because for $p = (a_0, a_1) \in P_{\Delta\,\mathrm{sc}_g}$, both $\mathrm{sc}(a_0, \mathcal{S}) < \mathrm{sc}(a_0, \mathcal{S}')$ and $\mathrm{sc}(a_1, \mathcal{S}) < \mathrm{sc}(a_1, \mathcal{S}')$. Therefore

$$\mathrm{sc}(p, \mathcal{S}) = \max(\mathrm{sc}(a_0, \mathcal{S}), \mathrm{sc}(a_1, \mathcal{S})) < \max(\mathrm{sc}(a_0, \mathcal{S}'), \mathrm{sc}(a_1, \mathcal{S}')) = \mathrm{sc}(p, \mathcal{S}')$$

Concluding $p \in P_{\Delta\,\mathrm{sc}}$.

The computation of the remaining elements of $P_{\Delta\,\mathrm{sc}}$ is more difficult. Those are the alternative pairs whose total score changes because of a change of $\mathrm{sc}_{dd}(a_i, \mathcal{S})$ for one of its alternatives. However, for an alternative pair $p = (a_0, a_1)$, $\mathrm{sc}_{dd}(a_0, \mathcal{S}) = \mathrm{sc}_{dd}(a_1, \mathcal{S})$ does not hold in general. Concluding, there might be alternative pairs for which $\mathrm{sc}_{dd}(a_i, \mathcal{S})$ increases but not $\mathrm{sc}(p, \mathcal{S})$. Nonetheless, for all remaining elements $p = (a_0, a_1) \in P_{\Delta\,\mathrm{sc}} \setminus P_{\Delta\,\mathrm{sc}_g}$, the destination delay score of at least one of the alternatives increases. We have

$$\mathrm{sc}(p, \mathcal{S}) \neq \mathrm{sc}(p, \mathcal{S}') \implies \mathrm{sc}_{dd}(a_0, \mathcal{S}) < \mathrm{sc}_{dd}(a_0, \mathcal{S}') \vee \mathrm{sc}_{dd}(a_1, \mathcal{S}) < \mathrm{sc}_{dd}(a_1, \mathcal{S}')$$

because of Equivalence 3.5. Consequently, we need to find all alternatives with $\mathrm{sc}_{dd}(a_i, \mathcal{S}) < \mathrm{sc}_{dd}(a_i, \mathcal{S}')$ efficiently. If we can do this, we can complete the construction of $P_{\Delta\,\mathrm{sc}}$ without considering every pair in P_{rem} individually. We have

$$\begin{aligned} \mathrm{sc}_{dd}(a_i, \mathcal{S}) &= \max_{e=(h,k)\in a_i} (d(e, \mathcal{S})) \\ &= \max_{e=(h,k)\in a_i} (\max_{c\in C}(d(e, c, \mathcal{S}))) \\ &= \max_{e=(h,k)\in a_i} (\max_{c\in C}(\mathrm{lp}_{\mathcal{S}}(h) + w(e, \mathrm{lp}_{\mathcal{S}}(h)) + \mathrm{lp}_{\mathcal{S}}(k, m_c) - \mathrm{lp}_{\emptyset}(m_c))) \\ &= \max_{e=(h,k)\in a_i} (\mathrm{lp}_{\mathcal{S}}(h) + w(e, \mathrm{lp}_{\mathcal{S}}(h)) + \max_{c\in C}(\mathrm{lp}_{\mathcal{S}}(k, m_c) - \mathrm{lp}_{\emptyset}(m_c))) \end{aligned}$$

Therefore, the destination delay score of an alternative may only increase if

$$d(e, \mathcal{S}) = \mathrm{lp}_{\mathcal{S}}(h) + w(e, \mathrm{lp}_{\mathcal{S}}(h)) + \max_{c\in C}(\mathrm{lp}_{\mathcal{S}}(k, m_c) - \mathrm{lp}_{\emptyset}(m_c))$$

increases for at least one of its edges. To check this, we need to evaluate $\max_{c\in C}(\mathrm{lp}_{\mathcal{S}}(k, m_c) - \mathrm{lp}_{\emptyset}(m_c))$ for multiple vertices k. Computing this value from scratch in every iteration is time-consuming. Therefore, we want to store and update it whenever we branch on a selection. We can do this using the same techniques as in Section 3.4.1. Therefore, we define $\mathrm{blp}_{\mathcal{S}}(k)$. It is the length of the longest path to k in $\tilde{G}(\mathcal{S})$. $\tilde{G}(\mathcal{S})$ is the transpose of $G(\mathcal{S})$. Additionally, $\tilde{G}(\mathcal{S})$ contains another vertex v_{bstart} that acts as the canonical start vertex for longest paths in $\tilde{G}(\mathcal{S})$. It is connected to the remainder of the graph by the fixed edges $e_c = (v_{bstart}, m_c)$ with weight $w(e_c) = -\mathrm{lp}_\emptyset(m_c)$. With those definitions in place, we have

$$\max_{c \in C}(\mathrm{lp}_{\mathcal{S}}(k, m_c) - \mathrm{lp}_\emptyset(m_c)) = \mathrm{blp}_{\mathcal{S}}(k)$$

We can update $\mathrm{blp}_{\mathcal{S}}(\cdot)$ by applying the same algorithm that we used in Section 3.4.1 to $\tilde{G}(\mathcal{S})$. Whenever we encounter a time-dependant edge (k, h) in this computation we use $\mathrm{lp}_{\mathcal{S}}(h)$ as the value for the time (e.g. $w((k,h)) = w((k,h), \mathrm{lp}_s el(h)))$[2].

Now, we can efficiently compute both $\mathrm{lp}_{\mathcal{S}}(\cdot)$ and $\mathrm{blp}_{\mathcal{S}}(\cdot)$. Therefore, it is easy to check whether

$$d(e, \mathcal{S}) = \mathrm{lp}_{\mathcal{S}}(h) + w(e, \mathrm{lp}_{\mathcal{S}}(h)) + \mathrm{blp}_{\mathcal{S}}(k) < \mathrm{lp}_{\mathcal{S}'}(h) + w(e, \mathrm{lp}_{\mathcal{S}'}(h)) + \mathrm{blp}_{\mathcal{S}'}(k) = d(e, \mathcal{S}')$$

for an alternative edge $e = (h, k)$. As discussed before, $\mathrm{lp}_{\mathcal{S}}(h) \leq \mathrm{lp}_{\mathcal{S}'}(h)$ and $\mathrm{blp}_{\mathcal{S}}(k) \leq \mathrm{blp}_{\mathcal{S}'}(k)$ for $\mathcal{S} \subseteq \mathcal{S}'$. Moreover, we constructed all time-dependent edges in a way that $\mathrm{lp}_{\mathcal{S}}(h) + w(e, \mathrm{lp}_{\mathcal{S}}(h)) \leq \mathrm{lp}_{\mathcal{S}'}(h) + w(e, \mathrm{lp}_{\mathcal{S}'}(h))$ (cf. Section 3.1.8). Furthermore, $w(e, \mathrm{lp}_{\mathcal{S}}(h))$ can only increase if $\mathrm{lp}_{\mathcal{S}}(h)$ increases. As a consequence, it suffices to look at all unselected edges (h, k) for which $\mathrm{lp}_{\mathcal{S}}(h) < \mathrm{lp}_{\mathcal{S}'}(h)$ or $\mathrm{blp}_{\mathcal{S}}(k) < \mathrm{blp}_{\mathcal{S}'}(k)$. We can efficiently find those by extending Algorithm 1 from Section 3.4.1. Before line 37 we add another loop. When *updateVertex* is called with the parameter v, the loop goes through all $e \in \{(v, w) | \exists (a_0, a_1) \in P_{rem} : (v, w) \in a_0 \vee (v, w) \in a_1\}$ and stores them in a list. Then, when the algorithm terminates, the list contains exactly all edges $e = (v, w)$ for which $\mathrm{lp}_{\mathcal{S}}(v) < \mathrm{lp}_{\mathcal{S}'}(v)$. We do the same when we apply the algorithm for the calculation of $\mathrm{blp}_{\mathcal{S}'}(\cdot)$. Thereby, we retrieve a list of all edges (v, w) for which $\mathrm{blp}_{\mathcal{S}}(w) < \mathrm{blp}_{\mathcal{S}'}(w)$[3]. Algorithm 2 shows the modified version. This way we can retrieve the relevant edges without an additional pass through the graph. Let E_Δ be the union of both retrieved lists.

To conclude our calculation, we check the following for every alternative edge $e \in E_\Delta$. First, let a be the alternative that e is contained in. Furthermore, let p be the alternative pair of a. If

$$\alpha \cdot d(e, \mathcal{S}') + \beta \cdot \mathrm{sc}_{ss}(a, \mathcal{S}') + \gamma \cdot \mathrm{sc}_g(a, \mathcal{S}') > \mathrm{sc}(p, \mathcal{S})$$

we know that $p \in P_{\Delta\,\mathrm{sc}}$. If not, the updated weights had no impact on the overall score of the alternative pair.

Finally, to compute $Q_{\mathrm{sc}}^{\mathcal{S}'}$, we increase the priority for all alternative pairs $p \in P_{\Delta\,\mathrm{sc}}$ to $\mathrm{sc}(p, \mathcal{S}')$.

3.4.4 Early pruning & memory reusing

Whenever we branch on a selection $\mathcal{S}$ in the branch-and-bound algorithm, we create two child selections $\mathcal{S}_0 = \mathcal{S} \cup a_0 \cup Stat^*(a_0)$ and $\mathcal{S}_1 = \mathcal{S} \cup a_1 \cup Stat^*(a_1)$ for an unselected alternative pair $p = (a_0, a_1)$. Therefore, with our current setup, we compute the metadata for both selections $\mathcal{S}_0$ and $\mathcal{S}_1$ by updating it from $\mathcal{S}$. After that, we calculate their lower bounds $\mathrm{LB}(\mathcal{S}_0)$ and $\mathrm{LB}(\mathcal{S}_1)$. Then, if one of the lower bounds is worse than

[2] Note that the edge (k, h) in $\tilde{G}(\mathcal{S})$ corresponds to the edge (h, k) in $G(\mathcal{S})$

[3] To be precise, we get their transposes in $\tilde{G}$. However, we are interested in the edges in the original graph

Algorithm 2 Adapted updateVertex Procedure

```
1:  procedure UPDATEVERTEX($v, \mathcal{S}$)
2:      if $\mathrm{lp}_{\mathcal{S}}(v) = -\infty$ then
3:          needsUpdate = needsUpdate $\setminus \{v\}$
4:          return 0
5:      end if
6:      for $(v, w) \in F \cup \mathcal{S}$ do
7:          relaxed $\leftarrow$ RELAXEDGE($(v, w)$)
8:          if relaxed $\wedge\ w \notin$ needsUpdate then
9:              needsUpdate $\leftarrow$ needsUpdate $\cup \{w\}$
10:         end if
11:     end for
12:     for $(v, w) \in \{(v, w) | \exists (a_0, a_1)$ not selected in $\mathcal{S} : (v, w) \in a_0 \vee (v, w) \in a_1\}$ do
13:         updatedEdges $\leftarrow$ updatedEdges $\cup \{(v, w)\}$
14:     end for
15:     needsUpdate $\leftarrow$ needsUpdate $\setminus \{v\}$
16: end procedure
```

the penalty of the current best solution, we prune the respective selection. In this case, we spend computation time updating the metadata without using (most of) it afterward. Moreover, we need to allocate memory for storing the additional metadata which comes with an additional time overhead. If we had known beforehand that one or both of the selections can be pruned, we could have saved the additional time. Therefore, we want to compute a lower bound for the child selection before updating the metadata. This lower bound will be naturally worse than $\mathrm{LB}(\mathcal{S}_i)$ as we have access to less information. However, if it is good enough to prune at least one of the child selections, we avoid the allocation of additional memory. Then, we can update the metadata of the parent selection in place and reuse it for the remaining, not pruned child selection. We can do this, as we do not need to access the parent selections metadata after branching on it. Therefore, we can modify it. By using this technique, we can process the selection of alternative pairs with at least one bad alternative faster.

We call the lower bound that we compute prior to updating the metadata $\mathrm{LB}^*(\mathcal{S})$. We define it to be the sum

$$\mathrm{LB}^*(\mathcal{S}) = \mathrm{LB}^*_{dd}(\mathcal{S}) + \mathrm{LB}^*_{ss}(\mathcal{S})$$

In the following, we explain the computation of its components

Distance penalty

First, we describe the computation of $\mathrm{LB}^*_{dd}(\cdot)$. Let w.l.o.g. $\mathcal{S}_0 = \mathcal{S} \cup a_0 \cup \mathit{Stat}^*(a_0)$ be the child selection that we consider. Then, we can define the set of new edges added to the graph $E_{new} = \mathcal{S} \setminus \mathcal{S}_0 = a_0 \cup \mathit{Stat}^*(a_0)$. For every $e = (i, j) \in E_{new}$ we can compute the following:

$$\begin{aligned}
dist_e &= \mathrm{lp}_{\mathcal{S}}(i) + w(e, \mathrm{lp}_{\mathcal{S}}(i)) + \mathrm{blp}_{\mathcal{S}}(j) + \mathrm{lp}_{\emptyset}(m_{c^*}) \\
&= \mathrm{lp}_{\mathcal{S}}(i) + w(e, \mathrm{lp}_{\mathcal{S}}(i)) + \max_{c \in C}(\mathrm{lp}_{\mathcal{S}}(k, m_c) - \mathrm{lp}_{\emptyset}(m_c)) + \mathrm{lp}_{\emptyset}(m_{c^*}) \\
&= \mathrm{lp}_{\mathcal{S}}(i) + w(e, \mathrm{lp}_{\mathcal{S}}(i)) + (\mathrm{lp}_{\mathcal{S}}(k, m_{c^*}) - \mathrm{lp}_{\emptyset}(m_{c^*})) + \mathrm{lp}_{\emptyset}(m_{c^*}) \\
&= \mathrm{lp}_{\mathcal{S}}(i) + w(e, \mathrm{lp}_{\mathcal{S}}(i)) + \mathrm{lp}_{\mathcal{S}}(k, m_{c^*})
\end{aligned}$$

Note that every component of the above sum is either part of the metadata stored for $\mathcal{S}$ or constant. The latter means that it can be computed upfront independently of a selection. The only additional information we need for the computation is the association between $\mathrm{blp}_{\mathcal{S}}(j) = \max_{c \in C}(\mathrm{lp}_{\mathcal{S}}(k, m_c) - \mathrm{lp}_{\emptyset}(m_c))$ and $c^* = \arg\max_{c \in C}(\mathrm{lp}_{\mathcal{S}}(k, m_c) - \mathrm{lp}_{\emptyset}(m_c))$. We can store this association as part of the metadata without much additional overhead. We compute and update it whenever we update $\mathrm{blp}(\cdot)$. For a vertex v, let $m(v) = \arg\max_{c \in C}(\mathrm{lp}_{\mathcal{S}}(v, m_c) - \mathrm{lp}_{\emptyset}(m_c))$ be this association. Now, $dist_e$ is the distance of the longest path from v_{start} to $m_{m(j)}$ in $G(\mathcal{S} \cup e)$ that uses e. As $e \in \mathcal{S}_0$, by definition $dist_e \leq \mathrm{lp}_{\mathcal{S}_0}(m_{m(j)})$. Let

$$p(d) = \begin{cases} 0 & \text{if } d < 180 \\ d \cdot \frac{125}{60} & \text{if } d \geq 180 \end{cases}$$

be the function that extracts the delay penalty from a delay. Furthermore, for $c \in C$ let

$$dist_c = \max(\{dist_e | e = (i, j) \in E_{new} \wedge m(j) = c\})$$

be the maximal distance to m_c that we can extract from all $dist_e$ values. Then, we can compute $\mathrm{LB}^*_{dd}(\mathcal{S}_0)$ as follows

$$\begin{aligned}
\mathrm{LB}^*_{dd}(\mathcal{S}_0) &= LB_{dd}(\mathcal{S}) + \sum_{\substack{m_c \in M \\ dist_c > \mathrm{lp}_{\mathcal{S}}(m_c)}} p(dist_c) - p(\mathrm{lp}_{\mathcal{S}}(m_c)) \\
&= LB_{dd}(\mathcal{S}) + \sum_{m_c \in M} p(\max(dist_c, \mathrm{lp}_{\mathcal{S}}(m_c))) - p(\mathrm{lp}_{\mathcal{S}}(m_c)) \\
&\leq \mathrm{LB}_{dd}(\mathcal{S}) + \sum_{m_c \in M} p(\mathrm{lp}_{\mathcal{S}_0}(m_c)) - p(\mathrm{lp}_{\mathcal{S}}(m_c)) \\
&\leq \mathrm{LB}_{dd}(\mathcal{S}_0)
\end{aligned}$$

Note that $\mathrm{LB}^*_{dd}(\mathcal{S}_0) \gg \mathrm{LB}_{dd}(\mathcal{S})$ if there are single edges $e = (i, j) \in E_{new}$ with $dist_e \gg \mathrm{lp}_{\mathcal{S}}(m_{m(j)})$. Therefore, the existence of such an edge makes LB^*_{dd} efficient for early pruning.

Skipped stop penalty

Second, we explain the computation of $\mathrm{LB}^*_{ss}(\cdot)$. Let again w.l.o.g. $\mathcal{S}_0 = \mathcal{S} \cup a_0 \cup Stat^*(a_0)$ be the child selection that we consider. Note that $\mathrm{LB}^*_{ss}(\mathcal{S}_0)$ can only increase compared to $\mathrm{LB}_{ss}(\mathcal{S})$ if either a_0 or any of its statically implied alternatives belong to a stop/pass alternative pair. If this is not the case, we set $\mathrm{LB}^*_{ss}(\mathcal{S}_0) = \mathrm{LB}_{ss}(\mathcal{S})$. We did not define any static implications that include stop/pass alternative pairs. Therefore, we can assume that in the other case $p = (a_0, a_1)$ is a stop/pass alternative pair and $Stat^*(a_0) = \emptyset$. Let si be the schedule item related to p. Moreover, let $c = \mathrm{course}(si)$ be the corresponding course. Again, if $\mathrm{ccat}(c) \neq Passenger$, we set $\mathrm{LB}^*_{ss}(\mathcal{S}_0) = \mathrm{LB}_{ss}(\mathcal{S})$ as p has no impact on the skipped stop penalty. If $\mathrm{ccat}(c) = Passenger$,

Condition	$\mathrm{LB}^*_{ss}(\mathcal{S}_0)$
p not a stop/pass alternative pair	$\mathrm{LB}_{ss}(\mathcal{S})$
$\mathrm{ccat}(c) \neq \mathit{Passenger}$	$\mathrm{LB}_{ss}(\mathcal{S})$
$a_0 = a_{stop} \wedge \mathrm{bsv}(si) > (\mathit{potBSV}_{c,\mathcal{S}})_1$	$\mathrm{LB}_{ss}(\mathcal{S})$
$a_0 = a_{pass} \wedge$ Equation 3.7	$\mathrm{LB}_{ss}(\mathcal{S}) + \mathrm{bsv}(si)$
otherwise	$\mathrm{LB}_{ss}(\mathcal{S}) + \mathrm{LB}_{ss,c}(\mathcal{S}_0) - \mathrm{LB}_{ss,c}(\mathcal{S})$

TABLE 3.1: Value of $\mathrm{LB}^*_{ss}(\mathcal{S}_0)$ based on conditions. The conditions must be checked from top to bottom.

there are multiple possibilities. First, let a_0 be the alternative with the semantics that c stops at node(si). Then, if $\mathrm{bsv}(si) > (\mathit{potBSV}_{c,\mathcal{S}})_1$, it does not have an impact on $\mathrm{LB}_{ss}(\cdot)$ (cf. Section 3.3.1). Therefore, we set $\mathrm{LB}^*_{ss}(\mathcal{S}_0) = \mathrm{LB}_{ss}(\mathcal{S}) = \mathrm{LB}_{ss}(\mathcal{S}_0)$. If this is not the case, we set

$$\mathrm{LB}^*_{ss}(\mathcal{S}_0) = \mathrm{LB}_{ss}(\mathcal{S}) + \mathrm{LB}_{ss,c}(\mathcal{S}_0) - \mathrm{LB}_{ss,c}(\mathcal{S}) \tag{3.6}$$

For this computation, we compute $\mathrm{LB}_{ss,c}(\mathcal{S}_0)$ by partially updating the relevant metadata. The concrete update process was described in Section 3.4.2.

Now, assume that a_0 is the alternative with the semantics that c passes at node(si). If

$$\mathrm{bsv}(si) > (\mathit{potBSV}_{c,\mathcal{S}})_1 \wedge \mathrm{bsv}(si) > \mathit{scSmlst}_{c,\mathcal{S}} \tag{3.7}$$

we set

$$\mathrm{LB}^*_{ss}(\mathcal{S}_0) = \mathrm{LB}_{ss}(\mathcal{S}) + \mathrm{bsv}(si)$$

This is possible, as the additional skipped stop does not have an impact on the first two summands of Sums 3.1 to 3.4. Rather, it will be part of the third summand. Therefore, independent of the sum chosen to compute $\mathrm{LB}_{ss,c}(\mathcal{S})$, $\mathrm{LB}_{ss,c}(\mathcal{S}_0)$ will be bigger by $\mathrm{bsv}(si)$. If Inequality 3.7 does not hold, we compute $\mathrm{LB}^*_{ss}(\mathcal{S}_0)$ as in Equation 3.6. Table 3.1 gives an overview of the potential values for $\mathrm{LB}^*_{ss}(\mathcal{S}_0)$.

3.4.5 Saving memory

During the run of the branch-and-bound algorithm, the number of selections on the stack can become quite big. This can become a problem, as we store a lot of metadata for each selection $\mathcal{S}$ on the stack. As discussed in the last sections, those are a lookup table for both $\mathrm{lp}_{\mathcal{S}}(\cdot)$ and $\mathrm{blp}_{\mathcal{S}}(\cdot)$ as well as the score queue $Q^{\mathcal{S}}_{\mathrm{sc}}$ and the value necessary for computing $\mathrm{LB}_{ss}(\mathcal{S})$. For a large enough stack and alternative graph, this data may exceed the size of a machine's main memory. This causes the operating system to swap out some of the data. This significantly slows down the entire computation as the access times of e.g. a hard drive are multiple orders of magnitude higher than the access time of the main memory. To avoid this, we want to limit the amount of main memory allocated by our algorithm.

We introduce a new parameter μ. We only keep the metadata of the topmost μ selection on the stack. Whenever we add a new selection to the stack, the metadata of the $(\mu + 1)$-th selection is deleted. We only keep the selection itself and its lower bound value $\mathrm{LB}_{total}(\mathcal{S})$ in memory. The latter is required for eagerly pruning any elements on the stack in case we find a new best selection (cf. Section 3.3.3). Now, whenever we extract a selection from the stack that does not have any metadata attached to it, we calculate its metadata from scratch. This way, we free the parts of the main memory that we are least likely to use again instead of the arbitrary parts swapped out by the operating system.

3.5 Speed-up techniques

The alternative graph produced by our approach becomes quite large for inputs with sizes comparable to the 2022 RAS problem. A large alternative graph affects the performance of the branch-and-bound algorithm in multiple ways. First, the search space considered by the algorithm increases with the number of alternative pairs. Therefore, it has to run for more iterations. Second, the computation time in every iteration scales with the number of nodes, edges and alternative pairs. This is because the runtime of the algorithms used to update a selection's metadata depends on those. Furthermore, the memory footprint of the metadata increases with a bigger graph. As a consequence, we try to reduce the size of the alternative graph to speed up the algorithm.

The remainder of this section is split into two parts. First, Section 3.5.1 presents a heuristic for predetermining choices during the construction of the graph. Then, Section 3.5.2 describes an approach for limiting the portion of the timetable that is considered at once. Both presented speed-up techniques are heuristics. Therefore, an algorithm using those techniques is not guaranteed to find the optimal solution any longer.

3.5.1 Predetermining alternatives

When creating the alternative graph, we consider all potential amendments. Therefore, in theory, the amended timetable can contain modified tracks in the afternoon for a problem that only contains incidents in the early morning. Also, our approach can modify the arrival and departure times in such a way that a course originally scheduled several hours after another course departs first at a node. Both scenarios are very unlikely to be part of the optimal solution. Nonetheless, we create alternative pairs, vertices, and edges for them that slow down the algorithm. To avoid this slowdown, we omit the alternative pairs in the first place. In the following, we explain how this can be done for the different types of alternative pairs. We start with stop/pass and track alternative pairs before continuing with headway alternative pairs.

Stop/Pass and track alternative pairs

For defining the stop/pass and track alternative pairs that we want to omit, we introduce the parameters lim_{sp} and lim_t. Now, when we build the alternative graph for a problem instance $\mathcal{P}$, we first calculate the time the last incident of $\mathcal{P}$ ends. We refer to this time as *liet* for **l**ast **i**ncident **e**nd **t**ime. Given $\mathcal{P} = (t, (RS_c)_{c\in C}, ERT, LD, ESD, ECD)$ it is

$$\begin{aligned} liet = \max(&\{t\} \cup \{e \mid (s,e,l,rt) \in ERT\} \\ &\cup \{e \mid (s,e,n,dt) \in ESD\} \\ &\cup \{\text{arrival}(si) + dt \mid (c,n,dt) \in ECD \land si \in S_c \land \text{node}(si) = n\} \\ &\cup \{\text{departure}((S_c)_0) + d \mid (c,d) \in LD\}) \end{aligned}$$

The formula defines the maximum end time of all incidents in the problem. For extended runtime and extended station dwell time incidents, we use the end time that is already given in their definition. For extended course dwell time incidents and late departures we calculate the expected end based on the original timetable. If there is no incident at all, we use the snapshot time as the end of the last incident.

Now, whenever we create a node operation n for a schedule item si, we check whether

$$\text{arrival}(si) - liet > lim_{sp}$$

If this is the case, we do not create a stop/pass alternative pair for this schedule item. Instead, we assume that the activity of the course at the node is predefined. As a result, we create fewer alternative pairs. Furthermore, we need fewer vertices and edges for modeling runtimes (cf. Section 3.1.4) and link headways (cf. Section 3.1.5). Symmetrically, we also check whether

$$\text{arrival}(si) - liet > lim_t$$

If this is the case, we do not create track alternative pairs. Rather, we set $T_n = \text{track}(si)$ (cf. Section 3.1.6). This implies that the track used by the course is the originally planned track from the schedule. Apart from the ommited alternative pair, we also create fewer vertices and edges for modeling track headway constraints because of this.

Headway alternative pairs

When omitting headway alternative pairs, we cannot ignore all alternative pairs after a certain threshold time. This would allow our model to produce invalid amended timetables. Rather, we want to avoid those alternative pairs for which the alternative that will be part of the optimal solution is predictable with high confidence. For example, let c_0 and c_1 be two courses that enter the same link. We assume that the original departure of c_0 is scheduled in the morning while the original departure of c_1 is in the evening. Then, we can say that it is very likely that c_1 also enters the link before c_0 in the optimal amended timetable. To avoid such alternative pairs in the graph, we add another parameter lim_{hw}. We now ommit any headways that have a planned difference between the courses of more than lim_{hw}. For example, let $c_0, c_1 \in C$ be two courses that use the same link ℓ. Let $si_0 \in S_{c_0}, si_1 \in S_{c_1}$ be the two schedule items describing the stop at the node just before the link. Let l_0, l_1 be the corresponding link operations. Also, let n_0, n_1 be the node operations preceding the link operations. Now, we check whether

$$|\text{departure}(si_0) - \text{departure}(si_1)| > lim_{hw} \tag{3.8}$$

If this is the case, we do not create the alternative pair $p_{l_0,l_1} = (a_0, a_1)$. However, if we simply omit it, our model can produce invalid results. To remedy this, let w.l.o.g. $\text{departure}(si_0) < \text{departure}(si_0)$. Then, we add the edges that would have been part of a_0 to F. Meaning, that we create fixed edges instead of alternative ones to still adhere to the required constraints. This technique does not reduce the number of nodes in the alternative graph but it reduces the number of alternative pairs and replaces alternative edges with fixed ones. The approach is similar to the modeling of headway constraints between realized and non-realized operations (cf. Section 3.1.9). In a way, we predetermine the chosen alternative while building the graph.

We do the same for track headway alternative pairs. However, here we check whether

$$|\text{arrival}(si_0) - \text{arrival}(si_1)| > lim_{hw} \tag{3.9}$$

Considering Incidents The current strategy for omitting headway alternative pairs does not consider the impact of incidents. For example, two courses might be scheduled sufficiently far apart in the original timetable to predefine the headway between them. However, due to a late incident, the predetermined alternative might be not as likely as it seems based on the original timetable. Therefore, we also want to consider potential incidents when predetermining headways. Before constructing any headway alternative edges, we calculate the longest paths $\text{lp}_\emptyset(v_{start}, o)$ to any operation o. Now, when deciding whether to omit headways we check whether

$$|\max(\text{departure}(si_0), \text{lp}_\emptyset(l_0)) - \max(\text{departure}(si_1), \text{lp}_\emptyset(l_1))| > lim_{hw}$$

holds instead of Inequality 3.8 or whether

$$|\max(\text{arrival}(si_0), \text{lp}_\emptyset(n_0)) - \max(\text{arrival}(si_1), \text{lp}_\emptyset(n_1))| > lim_{hw}$$

holds instead of Inequality 3.9. In both inequalities, we consider the maximum of the planned time of an operation and the minimal unconstrained time given all incidents.

Trivial static implications The predetermined alternatives statically imply other alternatives. However, we do not select them as part of the branch-and-bound algorithm but rather during the construction of the graph. Therefore, we do not consider those implications currently. We make up for this by introducing a set of trivially implied alternatives $\text{Stat}(\emptyset)$. Now, let $p = (a_0, a_1)$ be an alternative pair that we omitted by predetermining the alternative a_i. Let $\text{Stat}'(a_i)$ be the set of static implications of a_i in the graph where p is not omitted. Then, we add $\text{Stat}'(a_i)$ to $\text{Stat}(\emptyset)$. Now, at the beginning of the branch-and-bound algorithm, we do not start with the empty selection. Rather, we begin by adding all alternatives in $\text{Stat}(\emptyset)$. We also recursively add their respective static implications. We do the same thing when constructing the initial selection (cf. Section 3.3.4)

3.5.2 Limiting the extent

With our current model, we consider the entire remaining service day at once. This means that we create operations for courses scheduled in the evening even though the provided snapshot time is in the morning. However, in many cases, we want to fix problems in the schedule as soon as possible. Then, events in the morning do not impact the amended timetable in the afternoon. Nonetheless, we create a lot of vertices, edges and alternative pairs for operations far after the snapshot time. Those slow down the branch-and-bound algorithm. To avoid this, we want to limit the time interval that we consider at once. If we find a solution in the limited time interval such that all duties are on time at the end of the interval, we can complete this partial solution by using the original schedule from this point onward. If we do not find such a solution, we can iteratively solve an adapted problem. The adapted problem uses the amended timetable of the last iteration as its realized schedule. In the following, we describe the details of this iterated algorithm.

Let *extent* be a new input parameter for our algorithm. For a problem instance $\mathcal{P}$ with snapshot time t we define a threshold $thresh = t + extent$. Then, when creating the alternative graph, we only model courses that are planned to start before *thres*.

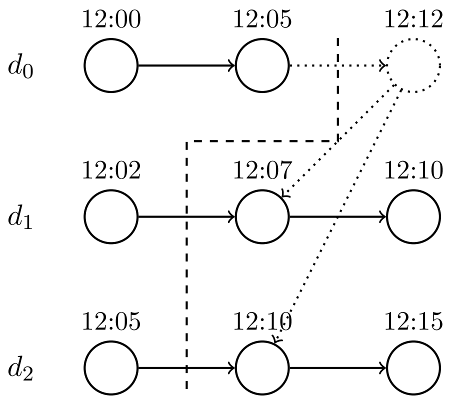


FIGURE 3.17: Simplified model of a partially amended timetable of three duties. The dotted vertices and edges represent operations and constraints that were not modeled in the current iteration. The dashed line shows the necessary cut-off point to get a valid schedule.

As a consequence, we reduce the number of modeled courses. We begin by solving this reduced problem. The result of this computation is an amended timetable $\mathcal{T}'$ that only contains a schedule for a subset of all courses. However, it may be invalid after a certain time τ. This is because we cannot ensure that we modeled all rolling stock duties up to the same point in time. The modeled schedule of some rolling stock duties goes further into the future than the schedule of others. We try to find the duty d that is not completed and ends first in our amended timetable. This means that the schedule of its last course is incomplete and that the time of its last schedule item[4] is minimal among all other last schedule items. Let τ be this time. The duty d may interfere with the other duties after τ. However, our model does not contain those potential interferences as we cut it off at some point. Figure 3.17 shows an example of those potential interferences. In the figure, $\tau = 12{:}05$. It is the time of the last operation of duty d_0. We can see, that the potential future operation of d_0 might have an impact on the already modeled operations of d_1 and d_2. In the concrete example, the time of the second operation of d_1 and d_2 is not valid. Therefore, the extracted amended schedule items are also not valid. To maintain the validity of the amended schedule, we have to cut off the resulting timetable at τ. We remove any schedule items whose arrival time is after τ. Also, for every remaining schedule item si with departure(si) $> \tau$ we set departure(si) to -1. The resulting cut-off timetable considers all constraints. In the figure, the cut-off point at the level of the alternative graph is shown as a dashed line.

Now, if $\mathcal{T}'$ ends on time for all duties and $liet < \tau$, we can complete the amended schedule with the schedule from the original timetable. This is an optimal completion as the original schedule does not cause any penalties. If not all duties end on time, we need to calculate the remaining timetable. We call our algorithm again with a modified problem instance. The modified problem instance uses τ as the new snapshot time. We use the amended schedule from the last iteration as the realized schedule of the new problem instance. We do this until $\mathcal{T}'$ no longer differs from $\mathcal{T}$ at τ or $\mathcal{T}'$ contains a complete schedule for all courses. Algorithm 3 shows a pseudo-code implementation of this iterated procedure.

[4]here time of the schedule item si means departure(si) if departure(si) $\neq -1$ and arrival(si) otherwise

Algorithm 3 Iterated approach for solving the complete problem

```
1: procedure ITERATEDRUN($\mathcal{I}, \mathcal{T}, \mathcal{P}$)
2:     $\mathcal{P}' \leftarrow \mathcal{P}$
3:     while $True$ do
4:         $G \leftarrow$ buildGraph($\mathcal{I}, \mathcal{T}, \mathcal{P}'$, $extend$)
5:         $\mathcal{T}' \leftarrow$ solve($G$)
6:         if isComplete($\mathcal{T}', \mathcal{T}$) then
7:             return $\mathcal{T}'$
8:         end if
9:         $t' \leftarrow$ firstDutyEnd($\mathcal{T}'$)
10:        $\mathcal{T}' \leftarrow$ pruneToThreshold($\mathcal{T}', t'$)
11:        if endsOnTime($\mathcal{T}', \mathcal{T}$) $and$ incidentsInPast($\mathcal{T}', \mathcal{P}'$) then
12:            return completeWithPlanned($\mathcal{T}', \mathcal{T}$)
13:        end if
14:        $\mathcal{P}' \leftarrow$ remainderAfterThreshold($\mathcal{P}', \mathcal{T}', t'$)
15:    end while
16: end procedure
```

Chapter 4

Evaluation

In this chapter, we evaluate our approach. For the evaluation, we use the infrastructure and timetable model supplied by the 2022 RAS Problem Solving Competition. We generate multiple sets of problem instances with different known future incidents. We solve each problem set multiple times with different configuration parameters for our algorithm.

The chapter starts with a brief overview of our general evaluation setup, the problem sets and the used parameters in Section 4.1. We continue with a short analysis of the impact of different parameters on the size of the generated alternative graph in Section 4.2. After that, we compare our approach with a first-come, first-served algorithm for finding optimal selections (Section 4.3). We further investigate the impact of different parameters on the calculated results in Section 4.4 before we continue with an analysis of the reasons for the increased runtime for some instances in Section 4.5. Section 4.6 finishes this chapter by presenting the results of our approach when applied to the example problems provided by the 2022 RAS Problem Solving Competition.

4.1 General information

4.1.1 Setup

We evaluated our implementation on a machine with an Intel Xeon E3-1246 v3 (4x 3.50GHz) CPU and 32GiB main memory. The application was written in C++ and compiled using GCC g++ (version 11.3) with the -O3 optimization flag. The source code is available at GitHub[1].

4.1.2 Model

The infrastructure and timetable model of the 2022 RAS Problem Solving Competition contains an entire service day of the London Elizabeth Line. Table 4.1 gives an overview of its size.

Entity	Nodes	Links	Rolling Stock Duties	Courses	Schedule Items
Count	104	216	91	1 259	28 324

TABLE 4.1: Table depicting the dimensions of the model

4.1.3 Problem instances

For the evaluation, we generated multiple sets of problem instances. Each set contains 20 problems. Their snapshot time t is uniformly distributed between 6:00 and 20:00.

[1] https://github.com/Emmeral/MetroDispositionOptimization

The realized schedule $(RS)_{c \in C}$ is the schedule $(S)_{c \in C}$ up to t. Each problem in the first problem set $\mathbb{P}_{ld}$ contains one late departure incident. The late departing course is randomly selected from the set of courses that are planned to depart after t. Additionally, the late departure may not be more than three hours after the snapshot time. The departure delay is chosen uniformly between five and 20 minutes. The second problem set $\mathbb{P}_{ert}$ contains problems with one extended runtime incident. The link of the incident is uniformly chosen from all links. The extended runtime lies between 150% and 300% of the link's maximal runtime. The interval of the incident is between 10 and 60 minutes long. It does not start before the snapshot time and ends at most three hours after it. The third problem set $\mathbb{P}_{edt}$ contains problems with one extended dwell time incident. The intervals were generated the same way as the intervals for the extended runtime incidents. The extended dwell time was uniformly chosen between one and three minutes. The last problem set $\mathbb{P}_{ecd}$ contains problem instances with one extended course dwell incident. The courses and nodes of the incident were randomly chosen such that the corresponding schedule item in the original timetable lies between the snapshot time and three hours after the snapshot time. The generated extended dwell time is between 30 seconds and 10 minutes longer than the originally planned dwell time. The final problem set $\mathbb{P}_{all}$ is the union of all prior defined problem sets.

$$\mathbb{P}_{all} = \mathbb{P}_{ld} \cup \mathbb{P}_{ert} \cup \mathbb{P}_{edt} \cup \mathbb{P}_{ecd}$$

Additionally, for evaluating properties of the generated alternative graphs, we define the problem $\mathcal{P}_{nothing}$. It does not contain any known future incidents. Its snapshot time t is 8:00. The realized schedule up to t corresponds to the original schedule.

Apart from the problem sets we generated ourselves, the 2022 RAS Problem Solving Competition also provides three example problems. Those are $\mathcal{P}_{incident\_abwdxr}$, $\mathcal{P}_{incident\_to\_heathrow}$, and $\mathcal{P}_{incident\_inside\_cos}$. Table 4.2 gives an overview of their specification.

Problem	t	Textual description
$\mathcal{P}_{incident\_abwdxr}$	18:15	Trains are departing from Abbey Wood terminal station with significant delays. This delay is forecasted to continue until approximately 18:30 with a *LD* incident.
$\mathcal{P}_{incident\_to\_heathrow}$	08:50	Trains heading to Heathrow Airport are running late. Moreover, an *ERT* incident is forecasted until 9:25, imposing an 8-minute runtime (instead of the planned three minute) between Heathrow tunnel junction and Heathrow terminal 2 & 3
$\mathcal{P}_{incident\_inside\_cos}$	08:00	Traffic inside the central part of the network is perturbed, with an *ESD* incident at Whitechapel between approx 8:00 and 9:00.

TABLE 4.2: Descriptions of provided problem instances. The textual description is copied from the 2022 RAS Problem Solving Competition [INF23]

4.1.4 Parameters

Throughout the evaluation, we run our algorithm with different parameters. If not stated otherwise explicitly, we use the default parameters listed in Table 4.3. Also, if not stated otherwise explicitly, we use the problem set $\mathbb{P}_{all}$.

Parameter	Default Value	Semantics
α	1	Weight for choice selection (cf. Section 3.3.2)
β	35	Weight for choice selection (cf. Section 3.3.2)
γ	10000	Weight for choice selection (cf. Section 3.3.2)
χ	9	Search strategy depth (cf. Section 3.3.3)
μ	1200	State metadata count (cf. Section 3.4.5)
lim_{sp}	60 min	Stop/Pass choice limit (cf. Section 3.5.1)
lim_t	60 min	Track choice limit (cf. Section 3.5.1)
lim_{hw}	40 min	Headway choice limit (cf. Section 3.5.1)
$extent$	24h	Extent of the constructed graph (cf. Section 3.5.2)

TABLE 4.3: Default evaluation parameters

Apart from setting those parameters, we terminate each application of the branch-and-bound algorithm after 30 minutes. This means that we choose the selection as a result that is the current best selection after 30 minutes of computation if the algorithm does not terminate before that. Note, that the total runtime can still be higher as we call the branch-and-bound algorithm multiple times for the same problem if the *extent* parameter is low (cf. Section 3.5.2).

4.2 Graph size

In this first section of the evaluation, we evaluate the impact of the limit parameters (lim_{sp}, lim_t and lim_{hw}) on the size of the constructed alternative graph. To achieve this, we build the alternative graph for the problem $\mathcal{P}_{nothing}$ with varying parameters. We do this to get an overview of the size of the alternative graph that has to be solved by our branch-and-bound approach.

We start by comparing the number of fixed edges, alternative edges and alternative pairs that are part of the constructed alternative graph for different limit parameters. The three plots of Figure 4.1 show the number of edges and alternative pairs relative to lim_{hw}. Note that the x-axis has a logarithmic scale. For each plot, we used different values for lim_{sp} and lim_t. We can see that with the maximum limits ($lim_{sp} = lim_t = lim_{hw} = 1024\text{min}$) the graph contains more than 26 million alternative edges grouped in roughly 4.7 million alternative pairs. Additionally, it has 307 534 fixed edges and 236 944 vertices. As expected, the number of alternative edges gets reduced by lowering lim_{hw}. However, this increases the number of fixed edges, as we predetermine certain choices this way. Nonetheless, the total number of edges is also reduced as not all alternative edges are substituted by fixed edges. Furthermore, we can see that the number of alternative edges is reduced if we reduce lim_{sp} and lim_t. However, the impact on the number of alternative pairs is not as significant. This is because the number of alternative pairs is only affected by the omitted stop/pass and track choices. Contrary to that, the number of alternative edges is additionally reduced by the fact that the omittance of stop/pass choices allows for modeling related link headway constraints with fewer edges (cf. Section 3.1.5). Similarly, the

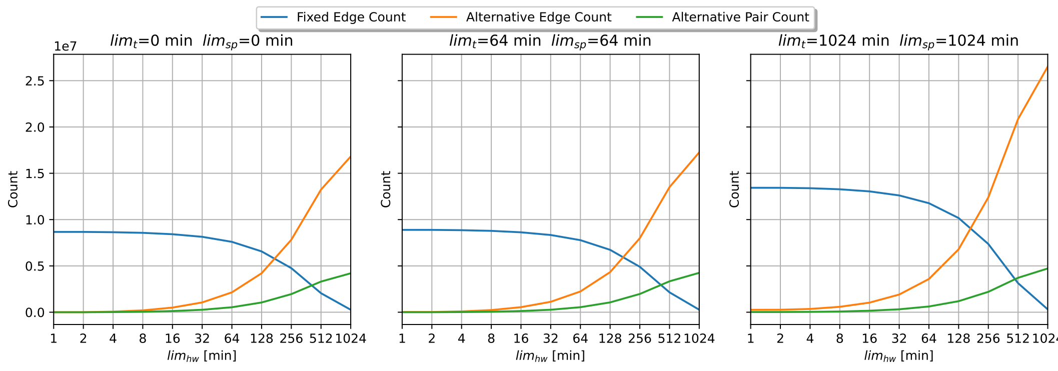
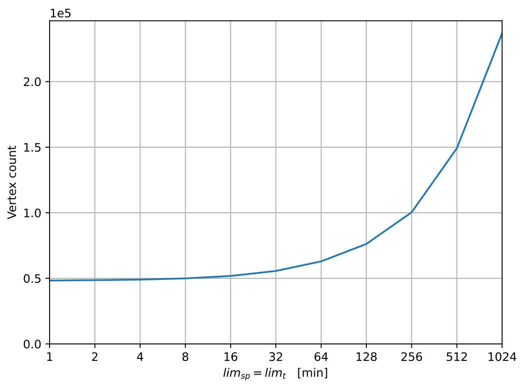


FIGURE 4.1: Number of fixed edges $|F|$, alternative edges $|\bigcup_{(a_0,a_1)\in A} a_0 \cup a_1|$ and alternative pairs $|A|$ in G with different values of lim_{sp}, lim_t and lim_{hw}



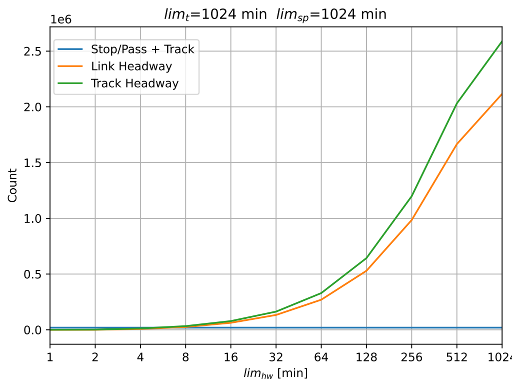


FIGURE 4.2: Number of vertices $|V|$ in G with different values of lim_{sp} and lim_t. lim_{hw} is fixed to 1024 minutes

FIGURE 4.3: Number of alternative pairs by type in G with different values of lim_{hw}. lim_{sp} and lim_t are fixed to 1024 minutes

omittance of track choices allows for modeling track headway constraints with fewer edges (cf. Section 3.1.6).

Figure 4.2 shows the impact of lim_{sp} and lim_t on the number of vertices in G. In the plot, lim_{sp} and lim_t are increased simultaneously. lim_{hw} is fixed to 1024 minutes. It does not have an impact on the number of vertices. As expected, we can see that increasing the limits increases the number of vertices. Nonetheless, the number of vertices is two orders of magnitude lower than the number of edges.

Lastly, Figure 4.3 shows the number of alternative pairs in the graph separated by their type. In the figure, lim_{sp} and lim_t are fixed to 1024 minutes. Because of this, we have a constant number of stop/pass alternative pairs (10 030) and track alternative pairs (9 392). As expected, the number of track and link headway alternative pairs grows when lim_{hw} is increased. Already at $lim_{hw} = 4$min there are roughly the same number of track (12 121) and link headway alternative pairs (8 218) as there are remaining alternative pairs. For higher headway limits, the number increases. Then, the number of headway alternative pairs is three orders of magnitude higher than the number of remaining alternative pairs. With $lim_{hw} = 1024$min we have 2 588 062 link headway alternative pairs and 2 116 107 track headway alternative pairs.

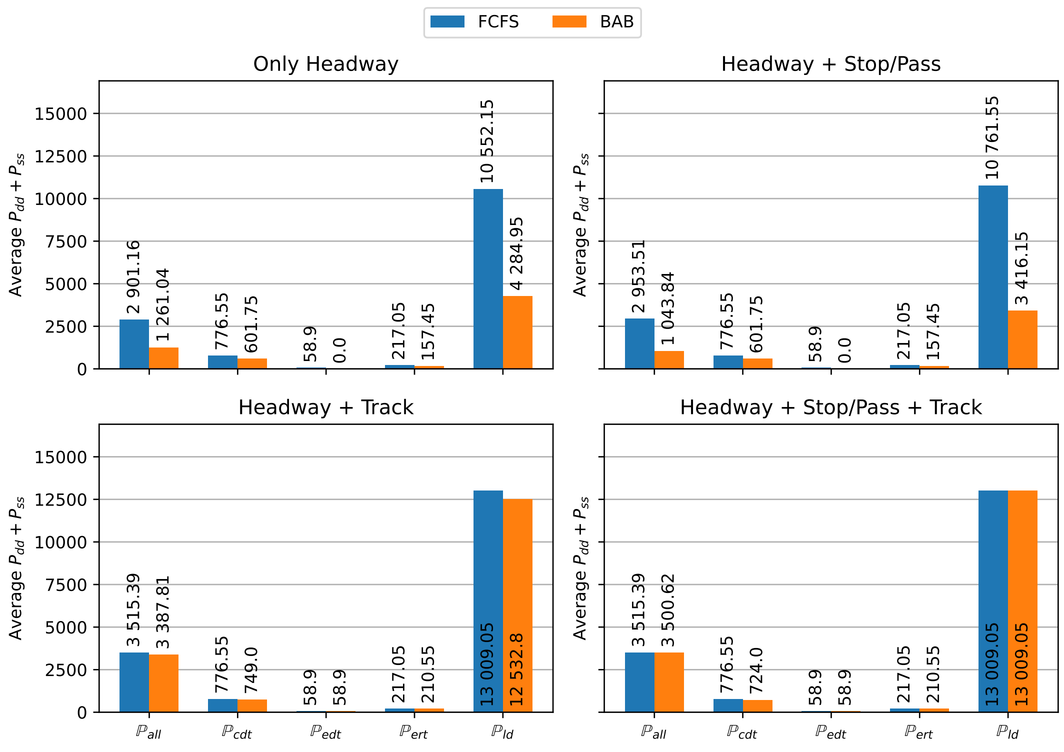


FIGURE 4.4: Comparison of the resulting penalty between the branch-and-bound (BAB) and first-come, first-served (FCFS) approach. Shown is the result for different parameters for lim_{sp}, lim_t.

4.3 Comparison to FCFS

In this section, we want to compare our branch-and-bound algorithm to a first-come, first-served (FCFS) approach. The FCFS algorithm is the same that is used to calculate the initial complete selection for the branch-and-bound algorithm (cf. Section 3.3.4). Instead of supplying it to the branch-and-bound algorithm, we use it directly to construct the amended timetable.

We ran the comparison with different parameter sets for building the graph. The concrete parameters are listed in Table 4.4. Note that the effect of the configuration "Only Headway" is that no stop/pass and track alternative pairs are created. Similarly, "Headway + Track" avoids the creation of stop/pass alternative pairs whereas "Headway + Stop/Pass" does not create track alternative pairs.

Name	Abbreviation	lim_{sp}	lim_t
Only Headway	HW	$-\infty$	$-\infty$
Headway + Stop/Pass	HW+SP	60 min	$-\infty$
Headway + Track	HW+T	$-\infty$	60 min
Headway + Stop/Pass + Track	HW+SP+T	60 min	60 min

TABLE 4.4: Parameters for penalty comparison evaluation

We ran the branch-and-bound and the first-come, first-served algorithm for each configuration. The results can be seen in Figure 4.4. The results are plotted individually for the different problem sets. The height of the bars represents the average sum of P_{dd} and P_{ss} across the amended timetables for all problems in the respective problem set. First, we can see that on average the penalty of the solution found by the

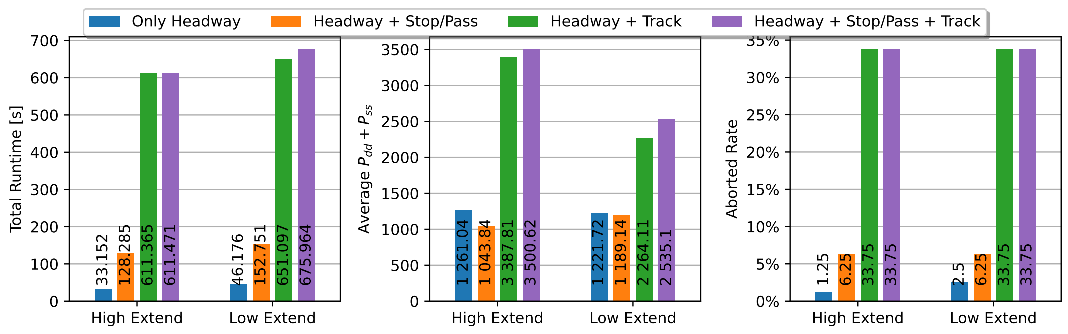


FIGURE 4.5: Comparison of the average penalty, runtime and abort percentage of the BAB approach with different parameters for *extent*, lim_{sp} and lim_t

BAB approach is better than the penalty of the solution found by the FCFS approach. This is no surprise, as the penalty of the FCFS approach acts as an upper bound for the BAB approach. However, we can also see that for the configurations with fewer degrees of freedom, the branch-and-bound approach achieves better results. Most notably, there is almost no improvement for the HW+SP+T configuration. Also, the improvements of the"HW+T configuration are minor. We discuss the reasons for that in Section 4.4. Nonetheless, the overall biggest improvement is that of the HW+SP configuration. For the combined problem set $\mathbb{P}_{all}$ the branch-and-bound approach achieves an average penalty of 1 043.84 whereas the average penalty of the FCFS approach is 2 953.51. Apart from that, we can see that $\mathbb{P}_{ld}$ seems to be the most difficult problem set for both the FCFS and the BAB approach. Its average penalty is 3.64 times higher than the average penalty for $\mathbb{P}_{all}$ for the FCFS approach and 3.27 times higher for the best BAB configuration. For all problems in the extended dwell time problem set $\mathbb{P}_{edt}$, the BAB approach with configuration HW and HW+SP computed an amended timetable with a penalty of 0. However, the average penalty of the solutions computed by the FCFS approach is also quite small (58.9).

4.4 Comparison between parameters

We now want to further investigate the difference in the results when we run the branch-and-bound algorithm against alternative graphs constructed with different parameters. In addition to the parameter sets introduced in the last chapter (cf. Table 4.4), we now also differentiate between different values for *extent*. We define "High Extent" to be the default *extent* value of 24 hours. "Low Extent" describes an *extent* value of three hours. We used $\mathbb{P}_{all}$ as the problem set for this evaluation.

Figure 4.5 shows the average runtimes, penalties and rate of aborted computations for all configurations. First, we want to compare the runtime and penalty of the two promising configurations (HW, HW+SP) between a low and high *extent* value. Interestingly, for both configurations, the average runtime with a low *extent* value is higher than the average runtime with a high *extent* value. Moreover, the penalty for the HW configuration with a low *extent* value is slightly lower than the penalty with a high *extent* value (1221.71 / 1261.04). For the HW+SP configuration, this is not the case. Here the penalty value for a low *extent* value is 13.9% higher than the penalty for the high *extent* value. We can also see that the average runtime for the configurations with track decisions (HW+T and HW+SP+T) is a lot higher than the

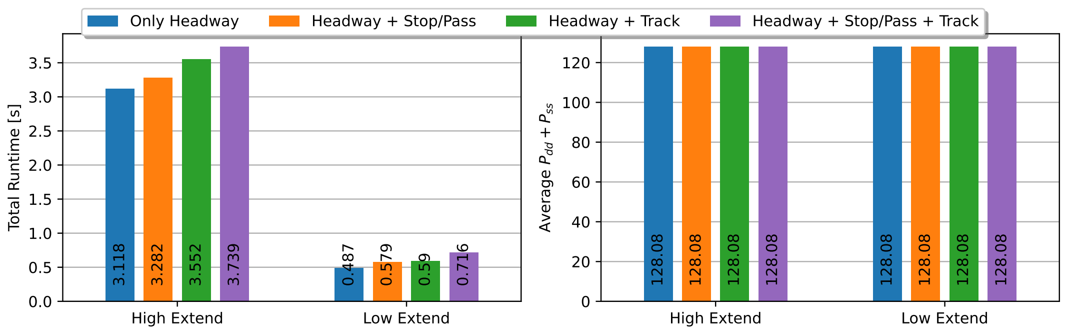


FIGURE 4.6: Comparison of the average penalty and runtime for problems for which non of the configurations caused the early termination of the BAB algorithm

average runtime for the other configurations. On average, both configurations need more than 10 minutes to finish.

The reasons for those observations can be seen in the rightmost plot of the figure. It shows the percentage of problems for which we had to terminate the branch-and-bound algorithm because the computation time reached the timeout of 30 minutes. We say that we *aborted* the algorithm. Note, that for the lower *extent* value there could be multiple branch-and-bound runs per problem instance. In this case, we mark the whole computation as aborted if at least one of the branch-and-bound runs reaches the timeout.

We can see that we have aborted runs for all configurations. Those explain why sometimes the penalty value for configurations with a low *extent* value is smaller than the penalty value for their counterpart with a high *extent* value. Once we terminate the algorithm, we can no longer guarantee that we found the best possible solution for the present configuration. Apart from that, we can see that for the configurations with track alternative pairs the percentage of aborted runs is way higher than for the other configurations. For both HW+T and HW+SP+T, 33.75% of all runs were aborted. As a consequence, the runtime for roughly one-third of the cases is higher than 1800 seconds. This explains the high average runtime for these configurations. In the remainder of this section, we want to focus our evaluation on the non-aborted cases. After that, in Section 4.5 we investigate the reasons for the high abort rates.

We restrict our comparison to the problems for which the branch-and-bound runs were not aborted for any configuration. Figure 4.6 shows the average runtime and penalty for those problem sets. We can see that the difference in average runtime between the configurations is less significant. We still have a small increase of runtime with more degrees of freedom but it is not as big as in the prior comparison. We can explain it by the increased complexity of the alternative graph. The average runtimes for a high *extent* value are all in an acceptable range for a real-time disposition system (3.118s (HW) to 3.739s (HW+SP+T)). Also, having a low *extent* value now positively affects the runtime for all configurations. Then, the average runtimes lie between 0.487s (HW) and 0.716s (HW+SP+T). Apart from the runtimes, the resulting penalty value is identical for all configurations independent of the *extent* value. This shows that the subset of considered problems in this analysis could not benefit from the additional possible amendments made possible by stop/pass and track decisions. Nonetheless, it also shows that the configurations with a low *extent* value still found the optimal solutions.

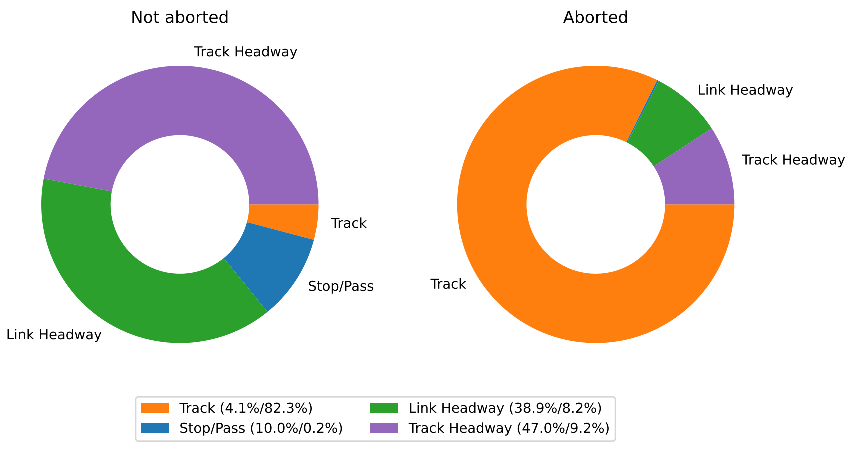


FIGURE 4.7: Distribution of considered choices split between problems for which the branch-and-bound algorithm aborted and those where it did not.

4.5 Investigating abort rate

We want to further analyze the difference between aborted and non-aborted problems. First, we are interested in the relative number of considered alternative pairs by type. This means, whenever we choose an alternative pair when branching on a selection in the branch-and-bound algorithm, we look at its type. The four possible types of alternative pairs are track, stop/pass, link headway and track headway. Then, we calculate the relative appearance of each alternative pair type in a single run of the branch-and-bound algorithm. We average those values across all problems. The averages are grouped by whether we aborted the branch-and-bound algorithm. Figure 4.7 shows the result of this analysis. We ran it with the high *extent* value and the HW+SP+T configuration. We can see that for the not-aborted problems, the majority of considered alternative pairs are either link or track headway alternative pairs. This is not surprising if we compare it with the overall distribution of alternative pair types in the graph (cf. Section 4.2 and Figure 4.3). The appearance of track and stop/pass alternative pairs is higher than in the overall distribution. However, this can be explained by the existence of static implications between headway alternative pairs. Because of those, multiple headway alternative pairs are considered at once. If we look at the aborted problems, the distribution is different. 82.3% of all considered alternative pairs are track alternative pairs. In contrast to that, on average, only 0.7% of the alternative pairs created for the aborted problems are track alternative pairs. This suggests that our algorithm has difficulties handling track alternative pairs.

To confirm this, we want to evaluate at which points our algorithm runs into dead ends in the search tree. Running into dead ends loses our algorithm computation time as we spend it exploring a wrong subtree. To gather information about dead ends, we collect the type distribution of the last alternative pairs that are added to selections whose children are both pruned immediately. In other words, during the algorithm, we take selections from the stack and build both of its child selections. Sometimes, the lower bounds of both child selections exceed the penalty value of the current best solution. In this case, both child selections are pruned immediately. We are now interested in the type of the last alternative pair that is added to the selection before this happens. This is helpful as it gives us an idea about where our lower bounds can be improved. This is because the lower bound value of the parent selection was

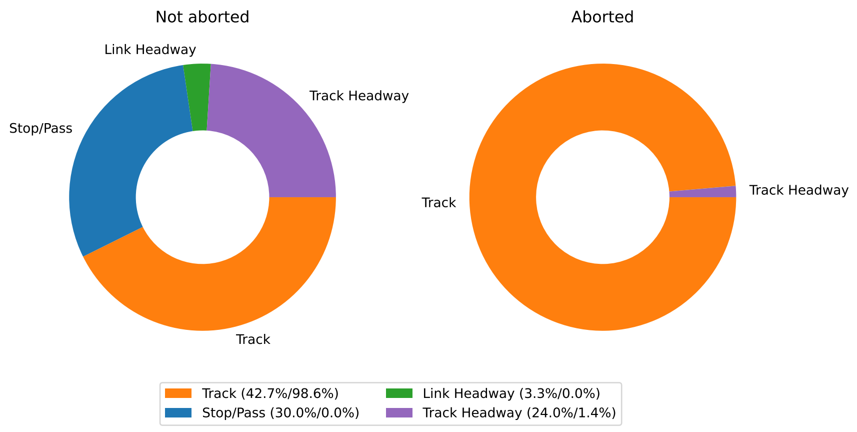


FIGURE 4.8: Distribution of the last alternative pair type before reaching a dead end in the search tree split between problems for which the branch-and-bound algorithm aborted and those where it did not.

not good enough to immediately prune it even though both of its child selections are pruned. We call such alternative pairs *dead-end* alternative pairs. Figure 4.8 gives an overview of their relative distribution per type split between aborted and not aborted problems.

We can see that for the not aborted problems, the dead-end alternative pairs are mostly split betwee track (42.7%), stop/pass (30.0%) and track headway (24.0%) alternative pairs. Track and stop/pass alternative pairs again appear more often than in the overall distribution of alternative pairs. For stop/pass alternative pairs, this could be because our lower bounds are the sum of lower bounds for the destination delay penalty LB_{dd} and lower bounds for the skipped stop penalty LB_{ss}. Now, when adding a stop/pass alternative pair $p = (a_{stop}, a_{pass})$ to a selection, one of the alternatives (a_{stop}) can increase LB_{dd} while the other (a_{pass}) can increase LB_{ss}. Therefore, both child selections might be invalid.

The result for the aborted problems is different. Here, 98.6% of all dead-end alternative pairs are track alternative pairs. This observation suggests that the branch-and-bound algorithm has difficulties finding the best track assignment for the different courses. This hypothesis is supported by the fact that the configurations without track alternative pairs have a better overall performance. The difficulties could be because the selection of track headway alternatives only has a full effect on LB_{dd} once the tracks of the related courses at the node are decided. Then, once this happens, both child selections might be invalid.

Next, we look at the number of times individual alternative pairs are added to the selections during the run of the algorithm. Therefore, for each run of the branch-and-bound algorithm, we collect the 20 individual alternative pairs that were added to selections the most. Together with them, we collect their type. We again plot the relative distribution of those types. It is weighted by the number of times the individual alternative pairs were considered. The results can be seen in Figure 4.9. For the aborted problems, on average, the most considered alternative pair type among the 20 most considered alternatives are track alternative pairs (83.7%). Given that track alternative pairs were already the most considered type for all alternative pairs (82.3% cf. Figure 4.7) this is not surprising. However, on average 82.1% of all considered alternative pairs are part of this top 20. Therefore, there is a relatively small subset of all alternative pairs that are considered more frequent than the remaining alternative pairs. Moreover, the overwhelming majority of those alternative pairs are

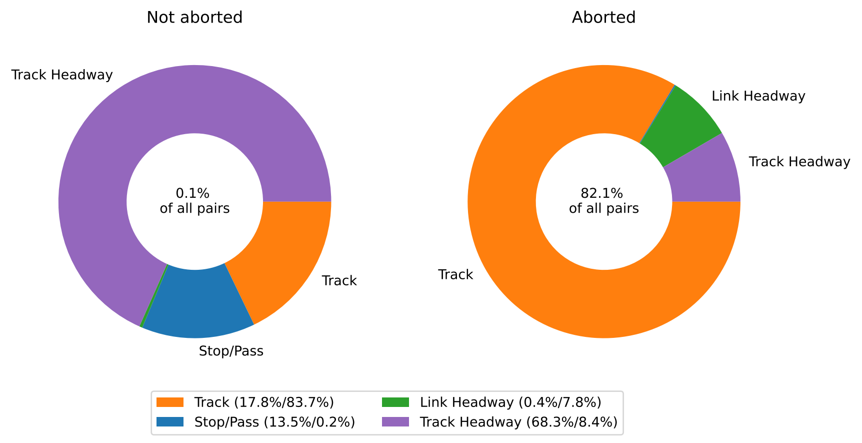


FIGURE 4.9: Distribution of the 20 most considered alternative pair types weighted by the number of considerations split between problems for which the branch and bound algorithm aborted and those for which it did not.

track alternative pairs. Concluding, there is a small number of track alternative pairs that use up the biggest share of the runtime of the algorithm. In comparison, for the not aborted problems, the 20 most considered alternative pairs make up only 0.1% of all considered alternative pairs. Additionally, track alternative pairs are not as dominant in this distribution. This observation further supports the hypothesis that a small subset of track decisions repeatably causes dead ends. Those dead ends are only revealed after other alternative pairs have been added to the selection. Then, the same track alternative pairs are added repeatably causing the pruning of both child selections. The search is continued somewhere higher in the search tree, repeating the whole procedure. However, the root alternative of this issue was the selection of an alternative pair even higher up in the search tree. Therefore, the algorithm will search through the entire subtree without pruning it early, causing high runtimes.

4.6 Evaluation of RAS example problems

In the final section of the evaluation, we want to analyze the results of our approach when applied to the example problems of the 2022 RAS Problem Solving Competition. For this analysis, we extended the time at which we terminate the branch-and-bound algorithm to 5 hours. We again used the same configurations as in the last sections with both a high and low *extent* value (cf. Table 4.4). The penalties of the computed amended timetables and the related runtimes are shown in Figures 4.10, 4.11 and 4.12 for $\mathcal{P}_{incident\_abwdxr}$, $\mathcal{P}_{incident\_to\_heathrow}$ and $\mathcal{P}_{incident\_inside\_cos}$ respectively.

For $\mathcal{P}_{incident\_abwdxr}$ we can see a consistent improvement compared to the FCFS approach. The amended timetable of the FCFS approach has a combined penalty of 22 868. The configurations without track alternative pairs achieve an improvement of 3701 independent of the value of the *extent* parameter. For configurations with track alternative pairs this improvement is even higher (4 111). Also, we do not abort the branch-and-bound algorithm for any configuration. The runtimes are in an acceptable range reaching from 1.62 seconds (HW, "Low Extent") to 49.02 seconds (HW+T, "High Extent"). Interestingly, the runtime for the HW+SP+T configuration is lower than the runtime for the HW+T configuration.

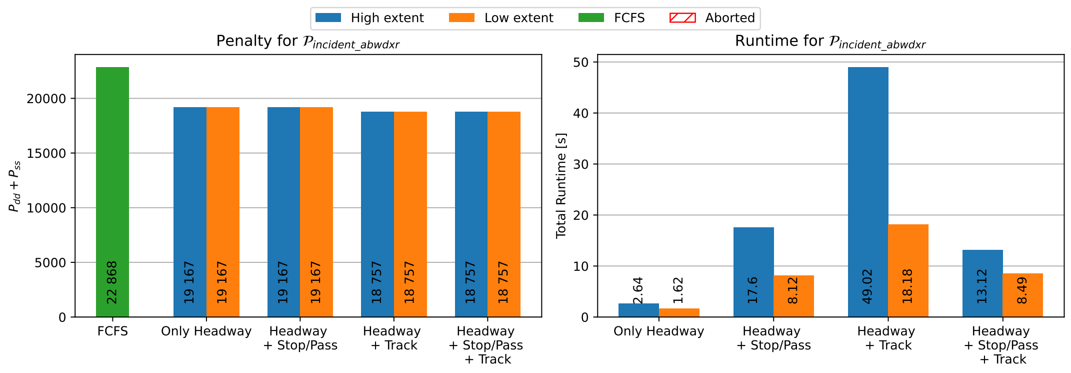


FIGURE 4.10: Comparison of the penalties and runtimes for $\mathcal{P}_{incident\_abwdxr}$ with different configurations. Striped bars mark aborted computations

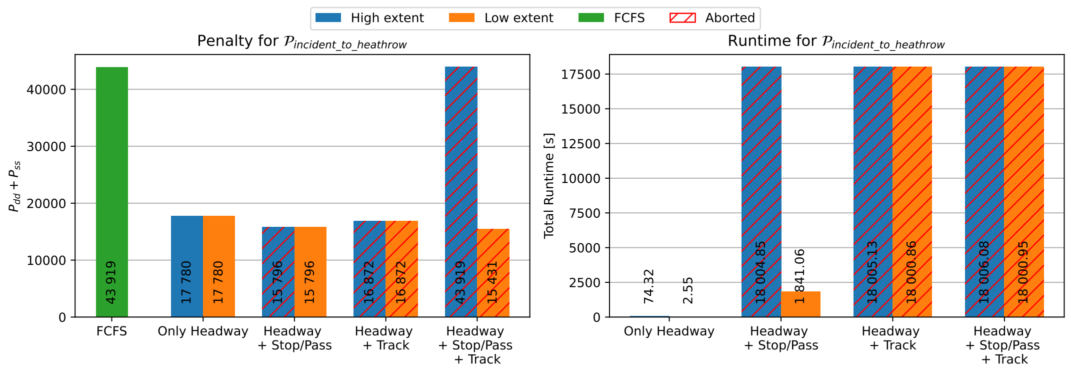


FIGURE 4.11: Comparison of the penalties and runtimes for $\mathcal{P}_{incident\_to\_heathrow}$ with different configurations. Striped bars mark aborted computations

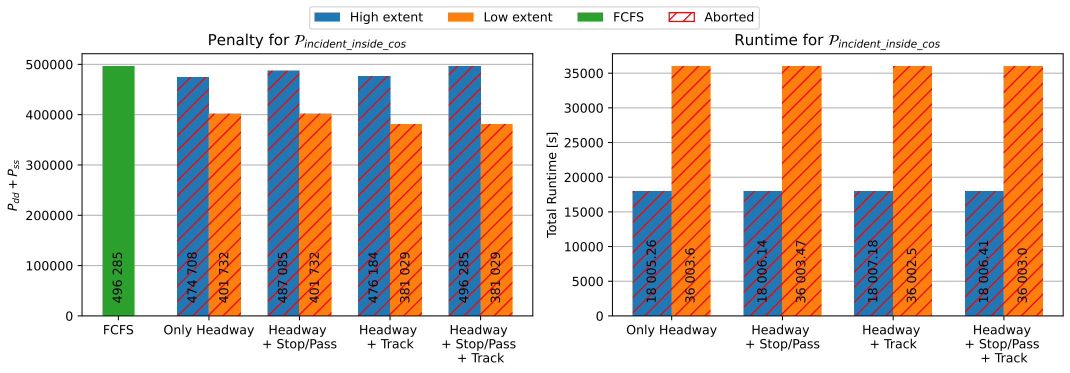


FIGURE 4.12: Comparison of the penalties and runtimes for $\mathcal{P}_{incident\_inside\_cos}$ with different configurations. Striped bars mark aborted computations

For $\mathcal{P}_{incident\_to\_heathrow}$, we get an penalty improvement for all but the HW+SP+T configuration with a high *extent* value. Surprisingly, the lowest penalty (15 431) is achieved by the same configuration with a low *extent* value. However, as the computation was aborted for both runs this is no contradiction. The other configurations also achieve penalty values of at most 17 780. This is a decrease of at least 59.5% compared to the FCFS value of 43 919. As expected, the runtime of the aborted cases is roughly 5 hours long. However, the HW configuration already terminates after 2.55 seconds with a low *extent* value and 74.32 seconds with a high *extent* value.

The last example problem $\mathcal{P}_{incident\_inside\_cos}$ is more difficult than the two prior problems. The penalty of the best solution found by the FCFS algorithm is 496 285. This is more than ten times higher than the FCFS penalty for the $\mathcal{P}_{incident\_to\_heathrow}$ problem. As a consequence, all runs of the branch-and-bound algorithm were aborted independent of the configuration. If we look at the runtimes, we can see that the configurations with a low *extent* value ran for roughly 10 hours. This is because the second run of the branch-and-bound algorithm for the modified problem timed out as well. Nonetheless, the configurations with a low *extent* value achieved better results. The lowest penalty is achieved by a low *extent* value and the HW+SP+T configuration. It is 381 029. Compared to that, the configurations with a high extent value only provide a minor improvement (HW and HW+T) or non at all ("Headway + Stop/Pass + Track"). The overall high penalty values suggest there is a major underlying incident. Therefore, we expect algorithms that also handle rolling stock duty amendments and course cancellations to perform better.

Chapter 5

Conclusion & Outlook

5.1 Conclusion

In this thesis, we introduced an alternative graph model for modeling a real-time train disposition problem. We extended an existing model to allow for more constraints and applied a modified branch-and-bound algorithm to optimize for a compound target function.

We started by introducing alternative graphs as a model for job-shop scheduling problems. Then, we defined the conflict resolution problem with fixed routes and showed how it can be solved using an alternative graph and a branch-and-bound algorithm. Additionally, we provided a formal description of the 2022 RAS problem.

We continued by adapting the alternative graph model for the conflict resolution problem to the 2022 RAS problem. We added dynamic stop/pass and track decisions and minimal run and headway times that depend on those. We also adapted and introduced new static implication rules for the modified model. Furthermore, we proposed a concrete branch-and-bound algorithm for optimizing the compound target function of the 2022 RAS problem. Thereby we introduced a dynamic state update process that avoids the repeated re-calculation of certain values accessed by the branch-and-bound algorithm. At last, we defined two heuristics for reducing the size of the generated alternative graphs.

We started the evaluation by confirming that the heuristics have a positive impact on the graph size. Then, we compared our approach to a first-come, first-served approach and found out that it achieves better results. We also saw that the addition of track decisions is the main driver of the increased runtime of the modified algorithm. We identified that the branch-and-bound runs that had to be aborted considered the same small subset of track alternative pairs repeatedly. Last, we analyzed our approach when applied to the example problems given by the 2022 RAS Problem Solving Competition. Again, we saw that it achieves better results than the FCFS algorithm. We also suspected that course cancellations and rolling stock amendments are necessary to find a satisfactory solution to the "Incident Inside COS" problem.

5.2 Outlook & future work

Better parameter evaluation

We fixed a lot of the algorithm's parameters for the evaluation. Additionally, the size of the used problem sets was limited. It would be interesting to run our approach with more parameter combinations to evaluate their respective impact on the overall performance and results. For example, we could evaluate the impact of changing lim_{sp}, lim_t and lim_{hw} on the overall runtime and penalty. Moreover, the impact of adapting μ would be another interesting evaluation as it changes the way the search space is explored. Also, running the algorithm with different factors for the score function (α, β, γ, cf Section 3.3.2) and estimating their effects on the performance of the algorithm is an option.

Track choice impact

We saw in the evaluation that track alternative pairs have a major impact on the performance of the algorithm. Moreover, a small portion of alternative pairs were chosen in the majority of iterations. Finding out the characteristics of the track alternative pairs that are responsible for the increased runtime might give additional insights to improve our proposed approach.

Apart from that, we could try to adapt the selection of the next alternative pair such that track alternative pairs are prioritized in some situations. For example, one could prioritize the related track alternative pairs whenever the usage of the same track by two courses would slow down one of the two. This would force the algorithm to select the tracks early. Then, the negative impacts of, for example, not enough tracks are directly visible in the destination delay lower bounds.

Additionally, one could try to find symmetries for the track selection. For example, consider a node with two available tracks in each direction. One could swap the used track for all courses and receive the same result in terms of the resulting penalty function. Therefore, the two selections are identical. However, the branch and bound algorithm still needs to prune both of them individually. By finding such and more complicated symmetries, the algorithm could be sped up further.

Combined lower bounds

Currently, we calculate the lower bounds of a selection $\mathrm{LB}(\mathcal{S})$ as the sum of the destination delay lower bounds $\mathrm{LB}_{dd}(\mathcal{S})$ and skipped stop lower bounds $\mathrm{LB}_{ss}(\mathcal{S})$ (cf. Section 3.3.1). This approach has the downside in that both components are considered individually. The skipped stops lower bounds consider the case in which no additional stops are skipped. The destination delay lower bounds assume that all remaining stops are skipped. Therefore, there might be no child selection at all that has a target function as low as this sum. By integrating the calculation of both lower bounds the total lower bound values could be improved.

Incorporating frequency

The current setup does not optimize for the frequency penalty P_f. In a future extension, it would be interesting to include this optimization. For that, one would need to calculate lower bounds for the frequency penalty. This is not trivial, as the order of multiple courses arriving at a node is not clear for an incomplete selection. Also, their arrival times can change drastically with the addition of new alternatives to the selection. Additionally, the frequency penalty can be improved by intentionally delaying the arrival at a node. Our model is not capable of mimicking such an intentional delay. As a consequence, we are missing a degree of freedom to fully optimize for this target.

Including more timetable amendments

We only allow for a subset of the allowed timetable amendments. As already discussed in Section 3.1.10 incorporating the additional amendments into the alternative graph blows up the size of the graph extensively. Nonetheless, the proposed additional amendments might help limit the overall delay. Therefore, a further investigation into how they can be integrated is desirable. One idea is to determine the rolling stock duty amendments in another step before applying the branch-and-bound algorithm. This is a similar approach to the one taken in [Cor+10]. However, one would need to adapt it to the additional constraints and more complicated objective function of the 2022 RAS problem. Moreover, one could investigate whether the inclusion of the track choices into this prior is desirable.

List of Figures

2.1 Job scheduling as graph 4
2.2 Job-Shop scheduling as graph 5
2.3 Blocking constraints in the alternative graph 6
2.4 Branch and bound search tree 7
2.5 Conflict resolution graph with same directions 10
2.6 Conflict resolution graph with different directions 12
2.7 London Elizabeth Line 13

3.1 Simple node and link graph 28
3.2 Start & Measure vertices and change end time 30
3.3 Stop/Pass alternative pairs 31
3.4 Variable runtimes graph 32
3.5 Simplified link headway constraint 33
3.6 Basic variable link headway constraint 34
3.7 Full variable link headway constraint 35
3.8 Fixed track headway constraints 37
3.9 Track dependent track headway 38
3.10 Extended dwell time visualization 40
3.11 Transition from realized operations 41
3.12 Static implications in same direction single-track 43
3.13 Static implications in same direction multi-track 44
3.14 Static implications in different directions 45
3.15 Static implications in same direction single-track 45
3.16 Branch-and-Bound search strategy 51
3.17 Pruning of partial amended timetable 63

4.1 Impact of lim_{sp}, lim_t and lim_{hw} on $|G|$ 68
4.2 Impact of lim_{sp}, lim_t and lim_{hw} on $|V|$ 68
4.3 Impact of lim_{hw} on alternative pair types 68
4.4 Comparison of penalty between BAB and FCFS 69
4.5 Comparison of penalty, runtime and abort percentage of BAB 70
4.6 Comparison of penalty, runtime for non-aborted cases 71
4.7 Distribution of considered choices 72
4.8 Distribution of the last choice before reaching a dead end 73
4.9 Distribution of the types of the 20 most considered alternative pairs 74
4.10 Penalties and runtimes for $\mathcal{P}_{incident\_abwdxr}$ 75
4.11 Penalties and runtimes for $\mathcal{P}_{incident\_to\_heathrow}$) 75
4.12 Penalties and runtimes for $\mathcal{P}_{incident\_inside\_cos}$ 76

List of Tables

2.1 CR static implications in same direction 11
2.2 CR static implications in opposite direction 12

3.1 Value of $\mathrm{LB}^*_{ss}(\mathcal{S}_0)$ based on conditions 59

4.1 Dimensions of the model 65
4.2 RAS example problem descriptions 66
4.3 Default evaluation parameters 67
4.4 Evaluation parameter sets 69

List of Algorithms

1 Longest path computation and update algorithm 53
2 Collection of updated edges 57
3 Iterated branch-and-bound algorithm 64

Bibliography

[BSV17] Andrea Bettinelli, Alberto Santini, and Daniele Vigo. "A real-time conflict solution algorithm for the train rescheduling problem". In: *Transportation Research Part B: Methodological* 106 (2017), pp. 237–265. ISSN: 0191-2615. DOI: https://doi.org/10.1016/j.trb.2017.10.005. URL: https://www.sciencedirect.com/science/article/pii/S0191261516301151.

[Cac+14] Valentina Cacchiani et al. "An overview of recovery models and algorithms for real-time railway rescheduling". In: *Transportation Research Part B: Methodological* 63 (2014), pp. 15–37. ISSN: 0191-2615. DOI: https://doi.org/10.1016/j.trb.2014.01.009. URL: https://www.sciencedirect.com/science/article/pii/S0191261514000198.

[Cor+10] Francesco Corman et al. "A tabu search algorithm for rerouting trains during rail operations". In: *Transportation Research Part B: Methodological* 44.1 (2010), pp. 175–192. ISSN: 0191-2615. DOI: https://doi.org/10.1016/j.trb.2009.05.004. URL: https://www.sciencedirect.com/science/article/pii/S0191261509000708.

[Cor+22] Thomas H. Cormen et al. *Introduction to Algorithms, 4th Edition.* MIT Press, 2022. ISBN: 978-0-262-04630-5. URL: http://mitpress.mit.edu/books/introduction-algorithms.

[DPP07] Andrea D'Ariano, Dario Pacciarelli, and Marco Pranzo. "A branch and bound algorithm for scheduling trains in a railway network". In: *Eur. J. Oper. Res.* 183.2 (2007), pp. 643–657. DOI: 10.1016/j.ejor.2006.10.034. URL: https://doi.org/10.1016/j.ejor.2006.10.034.

[INF23] INFORMS. *RAS Problem Repository.* 2023. URL: https://connect.informs.org/railway-applications/new-item3/problem-repository16 (visited on 05/23/2023).

[LW66] E. L. Lawler and D. E. Wood. "Branch-and-Bound Methods: A Survey". In: *Oper. Res.* 14.4 (1966), pp. 699–719. DOI: 10.1287/opre.14.4.699. URL: https://doi.org/10.1287/opre.14.4.699.

[MP02] Alessandro Mascis and Dario Pacciarelli. "Job-shop scheduling with blocking and no-wait constraints". In: *Eur. J. Oper. Res.* 143.3 (2002), pp. 498–517. DOI: 10.1016/S0377-2217(01)00338-1. URL: https://doi.org/10.1016/S0377-2217(01)00338-1.

[Pac02] Dario Pacciarelli. "Alternative graph formulation for solving complex factory-scheduling problems". In: *International Journal of Production Research* 40.15 (2002), pp. 3641–3653. DOI: 10.1080/00207540210136478. URL: https://doi.org/10.1080/00207540210136478.

[PK10] David J Pearce and Paul HJ Kelly. "A batch algorithm for maintaining a topological order". In: *Proceedings of the Thirty-Third Australasian Conferenc on Computer Science-Volume 102.* Citeseer, 2010, pp. 79–88.

[RS64] Bernard Roy and B Sussmann. “Les problemes dordonnancement avec contraintes disjonctives”. In: *Note ds* 9 (1964).

[SSJ18] Masoud Shakibayifar, Abdorreza Sheikholeslami, and Amin Jamili. “A Multi-Objective Decision Support System for Real-Time Train Rescheduling”. In: *IEEE Intell. Transp. Syst. Mag.* 10.3 (2018), pp. 94–109. DOI: 10.1109/MITS.2018.2842037. URL: https://doi.org/10.1109/MITS.2018.2842037.