Receding Horizon Control of Perturbed Railway Network Operation

Ate Conraad Kleijn

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Receding Horizon Control of Perturbed Railway Network Operation

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Ate Conraad Kleijn

May 15, 2012

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Railway networks, such as the one in the Netherlands, form an important means of transportation, both for passengers, as well as for transporting goods. Train services carrying passengers often run according to a predefined schedule or timetable. When those train services are delayed, for example due to accidents or malfunctioning rolling stock, the affected train may not be able to run according to schedule any longer. When the railway network is dense and hosts different kinds of services, such as local and intercity services, this initial delay is easily passed on to other train services in the network, due to different stopping patterns and drive speeds. Human dispatchers, possibly aided by computer systems, make temporary modifications to the way the network is used by the trains running in the region of the disturbance. However, due to the high complexity and the limited time, these decisions may be optimal for only the designated area of the dispatcher, but far from optimal from a network perspective. Therefore, a railway network operator could have a major benefit from a system able to compute globally optimal decisions in the case of disturbances.

This thesis is written as part of the development of such a system. Specifically, this project aims at applying Model Predictive Control (MPC) to railway networks. In MPC, a model of the system is used to predict the future behaviour of the system within a prediction horizon. The principle of receding horizon control is employed to compute an optimal future input sequence, such that a certain cost is minimized. This cost is the total delay in the network within the prediction horizon. The inputs of the controlled system are associated to the order of a train pair on a track. Train orders can be swapped at stations and junctions. As there are many of those points present in the network, swapping orders offers the most possibilities and is effective in a wide range of delay scenarios. Therefore, in this thesis only order swaps are considered. The system is modelled within a max-plus algebraic framework, which allows for a structured representation and systematic approach, where the latter is especially useful for future endeavours to exploit max-plus system theory, such that for example model reduction can be applied to the generally very large railway models.

The algorithm presented in this thesis forms the basis for an MPC algorithm for railway networks. The development of a tailor-made receding horizon control algorithm for railway networks, has not been carried out before. First an extension of the existing max-plus linear model is presented, such that a model which is uncertain in the parameters is obtained. This

model already contains controllable train orders. The problem of finding the optimal order swaps, such that the total delay in the network is minimal, can be written as a Mixed Integer Linear Programming problem. Several test cases were derived to highlight the various aspects of the receding horizon control algorithm, such as how it copes with different parameter estimations at various points in time. Through the use of a time-based control horizon, a significant reduction in computation time was achieved an provides a very good method to overcome the computational complexity encountered during optimal control of railway networks. Although the prediction horizon is defined in the discrete event domain, the algorithm is easily modified to also contain a time-based prediction horizon, allowing for more freedom in tuning of the prediction horizon and thus also computation times.

Table of Contents

List of Figures

List of Tables

Acknowledgements

My gratitude goes out to my supervisor, Dr. Ir. Ton J.J. van den Boom, for his support throughout the project. I would also like to thank Ir. Bart Kersbergen, who joined the team the last couple of months. Thanks for your useful feedback and thinking-along!

Delft, University of Technology
May 15, 2012

Ate Conraad Kleijn

Chapter 1

Introduction

The Dutch railway network is used by both passenger and freight trains. The passenger trains, for the largest part operated by the NS (Nederlandse Spoorwegen), run according to a predefined schedule. In case of a disturbance, such that, for example, a train is forced to drive slowly, a scheduled arrival or departure cannot can be delayed. Especially when the network hosts a high density of trains, the delayed train easily gets into conflict with other trains. In that case, dispatchers decide on what actions to take, such that nominal network operation can be regained as quickly as possible. However, due to the high complexity of this decision problem, often a solution is chosen which is good enough for the designated area of the dispatcher, but far from optimal from a network point of view. A decision may introduce new, unforeseen problems at another point in the network. Therefore, a railway network operator could have a major benefit from a system able to compute decisions which are optimal from a network point of view. This chapter presents an introduction into these kind of systems. First, some terminology relevant to this thesis is introduced in Section 1-1. Then, the process of bringing train lines into operation is briefly covered in Section 1-2. Typical dispatching actions are presented in Section 1-3, after which automation approaches to compute dispatching actions are summarized in Section 1-4. Finally, in Section 1-5, the project goals are defined.

1-1 Railway Terminology

Below, a short overview of terms related to railway networks are introduced. For passenger transportation in the Netherlands, usually three types of services are differentiated:

Local service A train running a local service stops at all stations along its route.

Regional service A train running a regional service stops at less stations than a local train. For example, a regional service connecting stations A and B makes less planned stops than a train running a local service between stations A and B.

Intercity service A train running an intercity service stops only at the major stations along its route.

A network hosting various types of services as mentioned above, is said to be *heterogeneous*, whereas networks hosting one type of service (like subway networks) are said to be *homogeneous*.

In the Netherlands, for each type of service different types of rolling stock is generally used. Railway networks consist of a variety of track types, but in this thesis only the following are considered:

Single track One track that can be used by trains travelling in both directions.

Double track Two tracks of which each track can only be used by trains travelling in one direction.

Overtaking track Extra piece of track parallel to primary track on which slower trains can wait to let faster trains pass.

These various types of tracks could merge or split at various points in the network, or they cross either physically or by utilizing a flyover, such that crossing trains cannot hinder each other. Stations usually consist of multiple platforms and, depending on the size of the station, a complex routing infrastructure is presents around a station. Furthermore, the network traffic is regulated through use of switches and traffic lights. In this thesis however, only junctions, stations and the tracks connecting these points are modeled. The individual platforms at stations are not considered. A station is merely seen as a node in the network, connecting multiple tracks. In fact, the capacity at stations is assumed to be unlimited. This means that a station can host an unlimited number of trains. For each train using a track, the following three parameters are considered:

Running time The time a train spends to traverse a track.

Dwell time The time a train resides at a station to ensure a connection to another train and to let passengers transfer.

Headway time The time between two subsequent trains on the same track needed to ensure safety. Usually based on physical properties of a train, like acceleration and deceleration limits.

The parameters mentioned above are those considered in obtaining the railway network model, which is presented in Section 2-1.

During operation of the network, temporary adjustments may be made to the schedule due planned maintenance, holidays or unforseen events, like rolling stock or infrastructure failure. The last example, temporary adjustments due to unforseen events, forms the basis of the re-scheduling procedure considered in this thesis and is covered in Section 1-3. However, to give

the interested reader an insight into the process of setting up train services, a short summary is presented next. Otherwise, this part can be skipped and the reader is invited to continue reading from Section 1-3 on.

The process of setting up a train service generally goes through the following three phases:

- Line planning
- Timetable design
- Resource allocation

Each of these phases will be treated separately next.

1-2 Line planning

In the process of line planning, decisions are made on what the starting and ending stations of train lines are, routes of trains, where train lines connect and at which stations the line stops [1]. This influences the choice of which type of service (and rolling stock) operates on a line. Decisions made in this phase already affect the performance of a network in terms of sensitivity to delays. Train lines connecting cities far apart decrease the number of transfers needed for passengers travelling between those cities, but delays can spread far over space and time [2]. Literature on line planning problems have for example proposed methods for maximizing passenger comfort, by maximizing the number of direct connections or by minimizing the number of transfers. Passenger behaviour presents a big complication in line planning, since this is hard to model.

1-2-1 Timetable design

The railway public transportation system in the Netherlands runs according to a predefined timetable, i.e. each train has a predefined arrival and departure time at specific points, such as stations, in the network. In the Dutch railway system this timetable repeats every hour, i.e. at least once every hour a train runs on a specific train line. The timetable is designed such that different train lines connect to each other, passengers have time to transfer to another line, safety on the tracks is guaranteed and possibly some other specifications such as deploying as many trains as possible on the tracks (while still respecting safety issues and other constraints). The decisions made during line planning do not imply that a feasible timetable can be constructed. Therefore, there are often several iterations between line planning and timetable design. Reliability is one of the predominant performance measures in railway performance [1]. When aiming to design reliable timetables, robustness is an important property. A robust timetable is capable of absorbing small delays due to unforeseen circumstances, without the need of intervention by railway dispatchers [1]. A timetable can have this absorbing property if it contains margins in running and dwell times [3]. These margins are referred to as *slack* or *buffer* times. Trains which have acquired some small delay can then still operate according to the timetable. More in-depth information and references related to timetabling can be found in [2].

1-2-2 Resource allocation

This phase deals with rolling stock and crew schedules. Allocation of rolling stock is based on quality of service, efficiency and robustness. Quality of service relates to the availability of sufficient capacity to transport the expected number of passengers. Robustness in this case relates to avoiding disturbing processes such as coupling and uncoupling of train units. Railway planners aim for a balanced rolling stock distribution over the network [4], [5], which means that at each station, enough and the correct type of rolling stock is present.

The problem of constructing a crew schedule consists of assigning crew members to a feasible duty (sequence of tasks). Feasibility relates for example to duty length and the order of tasks.

At a train station, trains have to be assigned tracks and platforms. This process is constrained by efficient passenger transfer, arrival and departure times as defined in the timetable, infrastructure (switches etc.), possible coupling or decoupling and safety aspects. Algorithms to find optimal routings, usually employ dominance techniques, i.e. techniques to reduce the problem size by only considering relevant sections of the railway infrastructure. More in-depth information and references related to resource allocation can be found in [2].

1-3 Delays and Dispatching Actions

Malfunctioning rolling stock, malfunctioning infrastructure, bad weather conditions, large alighting and boarding times of passengers, accidents, and so on, are all unforseen events that disrupt the intended operation of a railway network and could result in large delays. A delayed train might get into conflict with another train at a crossing or junction. This may happen at several points, possibly far apart, and other trains may thus be indirectly influenced by a disturbance due to interference with the 'originally' delayed train. Delays resulting from this interaction are referred to as *secondary* or *knock-on* delays. Secondary delays appear because of the shared use of the same infrastructure, rolling stock connections, transfers in crew schedules, passenger transfers, and so on. During disturbed operations, decisions may have to be made on how trains should move, both in space and time, in order to regain the intended operation is quickly as possible. Sometimes, the delays can be absorbed quickly enough due to sufficient slack or buffer time in the schedule, and no actions are necessary. In case of more severe disturbances re-scheduling of train movements is necessary. A number of dispatching actions can be undertaken to reduce the negative effects of disturbances. This is described next.

1-3-1 Dispatching actions aimed at reducing secondary delays

During disturbances, the predefined timetable may no longer be feasible and arrival and departure times need to be re-scheduled. Re-scheduling results in a new, temporary timetable. This timetable must respect safety constraints and is ideally optimal in the sense that it keeps some cost to a minimum. This cost can be e.g. the number of delayed trains or the total delay with respect to the predefined timetable. The schedule should converge to the predefined timetable as quickly as possible. Timetables are designed such that some train lines connect at certain stations to let passengers transfer from one line to the other. If one of

the connecting trains is delayed it may be better to break the connection to prevent the delay from propagating to other trains. The decision of whether or not a train should wait for a delayed connecting train is referred to as *delay management* in literature [3], [6]. The goal of delay management usually is to minimize some form of passenger delay, like e.g. the sum of all passenger delays [6], or the average delay of a passenger [7]. Finding an algorithm to solve the delay management problem in practical time is a big challenge, due to the complexity of the decision problem [6], [8], [9].

In most literature on delay management, only *wait-depart decisions* are considered, whereas [10] also considers *priority decisions*. More generally, one could consider all allowed dispatching actions, as usually performed by human dispatchers in practice. The following dispatching actions may be taken [3]:

Changing stopping patterns If two trains are of the same rolling stock type and it is impossible for the successive train to overtake the delayed train, stopping patterns may be switched. A regional train then for example becomes an intercity train and vice versa. This happens after passengers in both trains are informed, of course.

Inserting an on-time train A train is inserted at an intermediate station in the route of a delayed train, according to the scheduled arrival time of the delayed train at that station. When the delayed train reaches the intermediate station, this train is removed from the route.

Increasing residual capacity It may be necessary to decrease the capacity use of (a part of) the network, since due to a delayed train some infrastructural constraints are no longer met. Increasing capacity along a train line can be achieved through multiple actions. Canceling a departure from a terminal will increase the capacity along the entire route of the train. A down side of this is that this results dissatisfied passengers and a surplus of inventory at a certain station along the line and a deficit of inventory at the end of the line, which may result in the cancellation of a departure at the end station. Another measure is to cancel stops at stations with minor passenger loads and few connecting lines.

Short-turn a train A train route is shortened by letting the train turn around before it has arrived at its final station. This is referred to as short-turning a train. The short-turned train is then removed from the track and cannot affect other trains any longer.

Changing train speed profiles In [11] a re-scheduling method based on altering train speed profiles is used together with train re-ordering. The method relies on accurate monitoring of train speeds and locations to predict possible conflicts. Updated train speeds are communicated to train drivers in order to resolve those conflicts in real-time.

Changing dwell times Changing the dwell time at scheduled stops could compensate for the acquired delay during a trip [12].

Rerouting Due to a closed section of a track, the train line is (partly) rerouted over a different part of the network.

Swapping train order Suppose an intercity train is stuck behind a delayed regional train. The intercity train may then overtake the regional train at a station with multiple platforms in the same direction. In another case, a conflict may arise at a junction. Instead of maintaining the order (as predefined in the timetable), the delayed train gives priority to the other train such that this train is not hindered.

1-4 Automation Approaches

It is the task of a railway dispatcher to choose the appropriate action or a combination of actions listed above (more may exist), while ensuring safety. The quality of the chosen dispatching action for the disturbance in question grows with the experience of the dispatcher [13]. However, while in this way feasible solutions are produced, they may be far from optimal. Especially in dense networks with a high degree of synchronization, disturbances on one line easily affect multiple lines and the effects can have a long duration. Therefore, dispatchers would greatly benefit from automated systems which can propose and/or evaluate solutions [13]. Such automated system is usually referred to as a *Decision Support System* or DSS. At the Dutch railways the automatic route setting system ARI provides a support to dispatchers [12]. In operations research there is a lot of focus on finding algorithms to optimally reschedule trains in real-time. Optimal can mean minimal total passenger delay, minimal cost for the railway company, a trade-off between the latter two etc. However, complexity is a major issue and therefore these algorithms are mostly heuristic and present near-optimal solutions. Most proposed methods can find suboptimal solutions in practical time only for small parts of railway networks, like junctions [14], single lines [15] or parts of a network [11]. In [16] a detailed overview of proposed models and algorithms is presented.

1-4-1 Summary of related work

The Decision Support System (DSS) presented in [17] was implemented in the Asturian regional railway network in Spain in 1998. The rescheduling is done such that a minimal deviation from the original timetable results and is based on heuristic methods. The rescheduling problem is defined as optimally re-assigning train units within a predefined time interval to carry out these affected services. The objective function which is to be maximized during the optimization is based on the number of passengers, the expected delay a service suffers and the priority of that service. The expected delay is based on historical data. The solution results from searching a tree, in which each level in the tree corresponds to a service and each branch represents a train unit able to carry out that service. To be able to solve the problem in real-time, the algorithm first explores and reduces the solution space of the problem. This is done by means of a depth-first search in which a solution is discarded (by pruning branches) if that solution does not belong to the k best solutions. The system was implemented in 1998 in the Asturian regional railway network, serving as a DSS to train controllers. Based on experience of the users, the system could come up with useful solutions within 5 minutes.

The method proposed in [11] uses variation of train speed in its rescheduling scheme. The objective of the rescheduling method is to minimise the maximum delay due to conflicts. For the train speed profiles of all the trains in the network considered, potential conflicts between trains are predicted and resolved. This is done in the Conflict Detection and Resolution (CDR) phase of the algorithm, which uses an *alternative graph* [18] to model the train scheduling problem. Three different CDR algorithms are presented, being: simple dispatching rules simulating a human dispatcher, a greedy heuristic which in each step eliminates a scheduling alternative resulting in the largest delay, and a branch-and-bound algorithm. The found schedule is then fed to the train speed coordination part of the algorithm. In this part, a minimum distance headway is ensured, while keeping acceptable train speed profiles. Acceptable train speed profiles mean that the physical limitations like acceleration and deceleration limits are respected. The CDR and train speed coordination are performed iteratively, until a feasible solution is found. Computational tests based on the railway network around Schiphol in the Netherlands were carried out. The method performed better compared to situations where a human dispatcher made decisions.

ROMA (Railway traffic Optimization by Means of Alternative graphs) is a traffic management system proposed in [12] which uses two separate algorithms iteratively for sequencing and rerouting of trains in real time. First, infeasible routes due to blocking, caused by e.g. a defective train blocking a single track, are rerouted to obtain feasible new routes. The routing obtained in this step is referred to as the *default routing*. For the default routing a conflict-free train sequence is computed. As in [11], an alternative graph formulation is used to model the train scheduling problem. Two algorithms are tested for computing the new train sequence. One being a branch & bound algorithm, the other simulates the ARI traffic management system, as used by the Dutch railways. Then, for the computed sequence, a local rerouting is computed to possibly further improve the solution. The rerouting is limited to local modifications only, such as change of platforms within stations and is computed using a local-search algorithm. The re-sequencing and rerouting are performed iteratively until no better solution is found, or a time limit has been reached. In [19], a tabu-search algorithm has been proposed to solve the rerouting problem in ROMA, yielding both improvement in quality and computational time. Computational tests were carried out on practical size instances referring to the Dutch dispatching area between Utrecht and Den Bosch. The algorithm produces near-optimal solutions in short times. For example, after 20 seconds the new version of ROMA produces solutions up to more than 15% better than the previous version of ROMA does within 180 seconds.

In [20] the rescheduling problem is formulated as a Mixed Integer Linear Programming (MILP) problem and is solved using CPLEX. An event based model is developed in which each event is associated to a train requesting access to a segment. During rescheduling, the starting and ending times of these events may be altered. To investigate solution quality versus computational time, four rescheduling strategies differing in timetable modification possibilities were tested. Computational tests on a large part of the Swedish railway network with dense and heterogeneous traffic were carried out to test the different strategies. However, since in some cases the method could not find a solution fast enough for real-time applications, a method was developed using a greedy algorithm which finds a solution in 30 seconds [21]. The model was also extended to contain a more detailed description of stations which serve as junction points for different lines. Only a single delayed train is considered

In [5] a rescheduling method based on the rolling stock inventory at stations is presented.

During a perturbation in the schedule of a railway system, a rescheduling of trains (rolling stock) may lead to a inefficient distribution of rolling stock over the stations, such that expensive overnight re-allocation of trains is necessary. While railway dispatchers usually come up with modifications of rolling stock circulation to work around a disturbance relatively easy, they mostly end up with an off-balance (i.e. a deviation from a target inventory level of a certain rolling stock type at a certain station), such that the original schedule cannot immediately be re-established. The method presented in [5] aims at minimizing this off-balance in rolling stock and is one of the pioneering papers using this approach. Together with [4], the authors also explicitly model the shunting possibilities at the stations. Shunting is defined as the action necessary to perform a composition change of a train. A composition change refers to the coupling or uncoupling of train units. The main difference between [4] and [5] is the method of solving the problem. In [4] an MILP problem is obtained and solved using CPLEX, whereas [5] uses heuristic methods, employing experience of railroad dispatchers. The fact the method in [4] is used by the Dutch railway operator NS (Nederlandse Spoorwegen) since 2005 makes it an obvious candidate for comparison with the method in [5].

Related work as basis for this thesis

In [22], a model predictive controller computing the most effective re-scheduling actions is presented. This controller uses predictions of future departure and arrival times to determine a control sequence leading to a minimum total delay of all trains, while taking into account the cost of control actions. The control actions are restricted to only changing order of trains running on the same track. The scheduled railway network is modelled within a max-plus linear framework. They show that the control problem of allowing to change any number of train orders on a track yields an MILP.

During disturbances, the order of train departures may be altered to prevent accumulation of delays. Changing train order leads to a different model. Each train order results in a different operational *mode* and during re-scheduling the system *switches* between these modes. Each mode consists of a set of permutations. A permutation is an order swap of two trains scheduled to run on the same track. A binary control variable is associated to each possible permutation. The problem of finding the optimal set of integer event times and binary control variables, subject to the operational constraints, can be written into a Mixed Integer Linear Programming (MILP) problem. For this, the max-plus algebraic ϵ is replaced by a large negative number β, such that the max-plus algebraic permutations can be rewritten into a form in conventional algebra. This then results in a set of equations which is affine in the control variables.

Through examples using fictive data, the authors show that the proposed method indeed reduces delay significantly. They mention that the complexity of the MILP could be reduced by limiting the decision variables to only the relevant ones, by employing a delay propagation algorithm as proposed in [23].

1-5 Problem description and project goals

When a train is delayed it may as a consequence interfere with other trains in the railway network, which results in secondary delays. Changing the order of trains passing through at junctions or stations can reduce these secondary delays. Since stations and junctions are inherent to the basic infrastructure of a railway network and since regulating the order at those points is easily implemented, this is the focus of the research. The order swaps must be optimal, in that the total delay is minimized from a network point of view. Therefore, it is necessary to know the impact of decisions on future events, since current can affect events in the long run.

The controllable (i.e. allowing re-ordering of trains) railway network is represented using a max-plus linear system. Such a system allows for a structured representation of the various operational constraints of a railway network. Furthermore, analysis of a railway network can be carried out efficiently using max-plus system theory.

Optimal re-orderings are computed within a model predictive control (MPC) framework. It has already been shown that the problem of finding an optimal re-ordering of trains yields a Mixed Integer Linear Programme (MILP) [22]. The application of MPC to railway networks has not been carried out before. MPC is based on the principle of receding horizon control, however, no receding horizon control algorithm for railway networks is available yet.

Challenges

The scheduled railway network is a time-varying discrete event system. The model used gives all (expected) event times per cycle. A cycle is defined as the occurrence of all train runs as contained in one period of the timetable. A train run is completed when the train in question has arrived at the end-point of a track. At any time during one cycle, the system description may change due to a disturbance. Then, the optimal event times and train order must be re-computed, while respecting already occurred events and implemented decisions. This is visualized in Figure 1-1.

Project goals

The main goal of this thesis is:

Design and implementation (in MATLAB*) of a receding horizon algorithm which computes an optimal future sequence of trains over some event horizon, such that in the event of primary delays, the sum of all future delays is reduced to a minimum.*

The main goal is divided into two more specific subgoals:

1. *The max-plus linear system reflects the operation according to a predefined timetable. In the event of disturbances, this description is no longer valid and has to be updated with estimated parameters (running and dwell times). The system description can thus change within the same event horizon and the optimum has to be recomputed for every change in parameters, while respecting past decisions and events. The model predictive controller thus has to be able to compute new optimal events times and train orders*

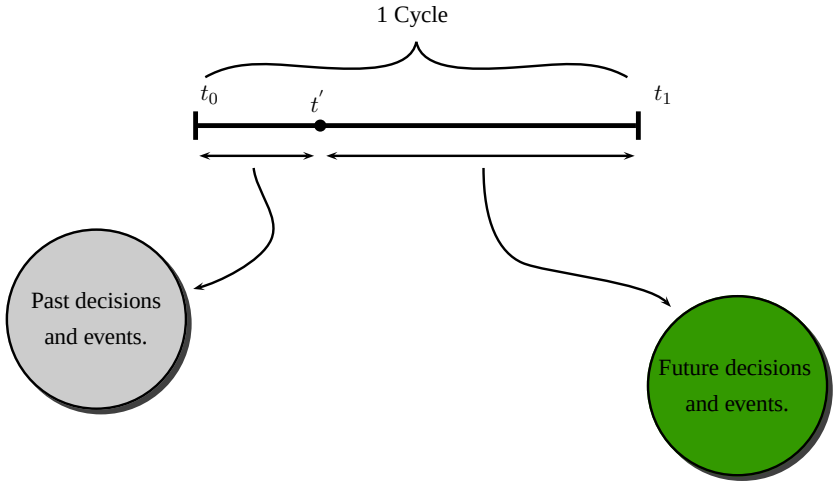

Figure 1-1: One cycle contains a set of past decisions and events and a set of future decisions and events. The controller is allowed to only optimize over the set of future decisions and events.

using an updated system description. Therefore, develop a receding horizon algorithm to deal with changes occurring both in the event and time domain.

2. *Investigate the effect of employing a control horizon on computational times and quality of solution. This control horizon is either defined over a fixed number of events within a variable time span or over a fixed time span containing a variable number of events.*

1-6 Thesis Outline

The following subjects are treated next:

Chapter 2: Modelling A structured railway network model based on max-plus algebra is presented here.

Chapter 3: Control This chapter presents how order swaps can be implemented in the model as presented in chapter 2. The model is structured further with respect to the inputs and a prediction model is presented. A receding horizon control strategy is defined for railway networks.

Chapter 4: Implementation The implementation of the receding horizon algorithm.

Chapter 5: Case Studies Case studies in which the receding horizon control algorithm is tested on a virtual railway network.

Chapter 6: Conclusions and Recommendations Discussion of the results obtained in chapter 5 and the main issues of receding horizon control of railway networks.

Chapter 2

Modelling

In this chapter an algebraic model describing the operation of railway networks [22] is presented. The relevant aspects of railway network operation are covered in Section 2-1, resulting in operational constraints which are captured in a max-plus algebraic system description. Max-plus algebra is introduced in Section 2-2. The max-plus algebraic model is introduced in Section 2-3, of which Section 2-3-2 presents a structured nominal system description, which is developed into a structured perturbed system description in Section 2-3-3.

2-1 Modelling Aspects

The previous chapter introduced the concept of minimizing train delays by changing the train order at different tracks. The controller to be designed will thus be able to change the train order at separate tracks. The endpoints of those tracks must give the possibility to change the train order. This is the case at stations and junctions. Therefore, a track is defined as the connection between two stations, two junctions or a station and a junction. A station is seen as a node connecting multiple tracks, with unlimited capacity.
Two types of junctions are present in the network. One junction merges two or more separate tracks into one (Figure 2-1) and at the other junction two tracks cross but do not merge (Figure 2-2).

On each track, trains can perform several activities, viz.:

- Traversing a track: a *running activity* with an associated *running time*.
- Waiting at a station: a *dwell* activity with an associated *dwell time*.

A running time is the time needed for a train to traverse a track (see Figure 2-3). The duration of a running activity differs for different types of trains. For example, the running

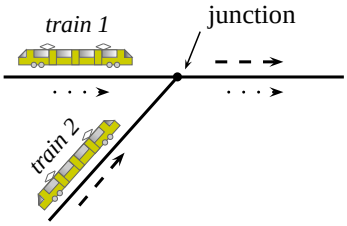

Figure 2-1: Junction merging two separate tracks into one.

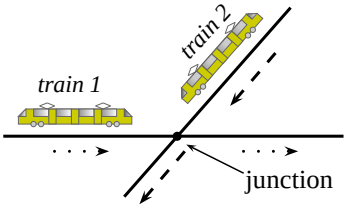

Figure 2-2: Junction at which tracks cross.

time of an intercity train will be smaller than that of a local train. A stop at a station is characterized by a dwell activity.

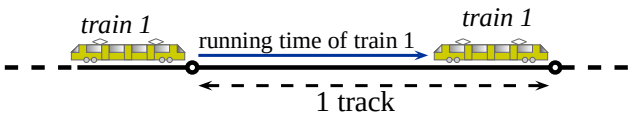

Figure 2-3: Illustration of a train run on a track.

For safety reasons, trains are not allowed to be too close to each other when sharing a track (see Figure 2-5). This separation time is called the *headway time*. The time difference between train 2 entering track 2 in Figure 2-5 and train 1 entering track 2 must at least be this headway time. Trains meeting at a junction as in Figure 2-1 are defined to be traveling in the same direction, whereas trains meeting at junctions like the one in Figure 2-2 are said to be traveling in opposite directions. However, since a junction is a point, a traveling direction cannot be well defined. Therefore, this junction is modeled as a *virtual* track. This virtual track is then given a running time equal to 0 minutes such that this track has the properties of a point. This representation is visualized in Figure 2-4. In general, the departure and arrival times of subsequent trains must be at least a headway time apart.

Each train has a departure and an arrival time for each track it visits. Moreover, each train has a *scheduled* departure and arrival time which will be obeyed in the disturbance-free case with a feasible timetable. Departures and arrivals are called *events*. These events are allowed to occur only in a specific order. For example, a train cannot arrive before it has departed and has traveled at least some time (running time) on a track. The minimum headway time dictates that a departure event of a train cannot occur if the departure event of the preceding train occurred less than a headway time earlier. Event occurrence thus determines the operation of the network. Due to this event-driven nature, a railway network

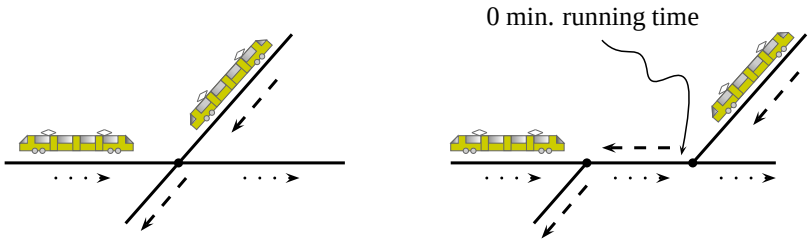

Figure 2-4: Trains crossing paths share a virtual track with 0 minutes running time.

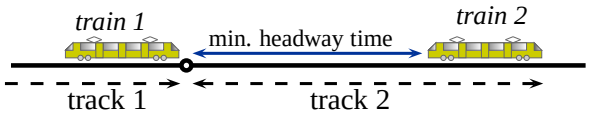

Figure 2-5: Illustration of headway time between two subsequent trains.

can be modeled as a *Discrete Event System* (DES). A DES is a discrete-state, event-driven system. Event-driven means that a discrete-state transition only happens in the occurrence of an asynchronous discrete event over time [24]. For example, a computer program changes from idle to busy (discrete state), when a user hits a button (discrete event). Modeling DES can be done with e.g. *automata* and *Petri nets* [24]. Railway networks belong to a special class of DES, viz. the class of DES's with synchronization and no concurrency. Events are synchronized as they are allowed to only occur as a consequence of other events. There is no concurrency, since for a feasible operation (e.g. according to the timetable) there will be no resource conflicts, i.e. there is one predefined order in which trains will use tracks (resources). This type of DES can be algebraically described by *max-plus algebra*, which will be the modeling formalism used in this thesis. Therefore, the next section is devoted to the basics of max-plus algebra.

2-2 Max Plus Algebra

This section introduces max-plus algebra and its properties as found in [25], which are relevant to this thesis. Basic concepts and definitions are introduced first, after which vector and matrix operations are defined over max-plus algebra.

2-2-1 Basic concepts and definitions

Define $\varepsilon = -\infty$ and $e = 0$, and define $\mathbb{R}_{\max}$ as the set $\mathbb{R} \cup \{\varepsilon\}$. Let $a, b \in \mathbb{R}_{\max}$ and define the operations $\oplus$ and $\otimes$ by

$$a \oplus b = \max(a, b) \quad \text{and} \quad a \otimes b = a + b. \tag{2-1}$$

For any $a \in \mathbb{R}_{\max}$ it holds that

$$a \oplus \varepsilon = \varepsilon \oplus a = a \tag{2-2}$$
$$a \otimes \varepsilon = \varepsilon \otimes a = \varepsilon \tag{2-3}$$

Max-plus algebraic multiplication ($\otimes$) precedes over max-plus algebraic addition ($\oplus$), as is the case in conventional algebra. Some max-plus algebraic properties are listed below:

- Associativity:
  $\forall x, y, z \in \mathbb{R}_{\max} : \quad x \oplus (y \oplus z) = (x \oplus y) \oplus z$
  $\forall x, y, z \in \mathbb{R}_{\max} : \quad x \otimes (y \otimes z) = (x \otimes y) \otimes z$
- Commutativity:
  $\forall x, y \in \mathbb{R}_{\max} : x \oplus y = y \oplus x$ and $x \otimes y = y \otimes x$
- Distributivity of $\otimes$ over $\oplus$:
  $\forall x, y, z \in \mathbb{R}_{\max} : \quad x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$
- Idem-potency of $\oplus$:
  $\forall x \in \mathbb{R}_{\max} : \quad x \oplus x = x$

2-2-2 Vectors and matrices

The elements of an $n \times m$ matrix A are defined over the set $\mathbb{R}_{\max}^{n \times m}$ and are denoted by

$$a_{ij} = [A]_{ij} \quad , \ 1 \leq i \leq n, \ 1 \leq j \leq m. \tag{2-4}$$

Matrix addition of $A \in \mathbb{R}_{\max}^{n \times m}$ and $B \in \mathbb{R}_{\max}^{n \times m}$, denoted by $A \oplus B$, is defined through

$$\begin{aligned} [A \oplus B]_{ij} &= a_{ij} \oplus b_{ij} \\ &= \max(a_{ij}, \ b_{ij}). \end{aligned} \tag{2-5}$$

Scalar multiplication of $\alpha \in \mathbb{R}_{\max}$ and $A \in \mathbb{R}_{\max}^{n \times m}$ is defined by

$$[\alpha \otimes A]_{ij} = \alpha \otimes a_{ij}. \tag{2-6}$$

The matrix product for matrices $A \in \mathbb{R}_{\max}^{n \times l}$ and $B \in \mathbb{R}_{\max}^{l \times m}$ is defined through

$$\begin{aligned} [A \otimes B]_{ik} &= \bigoplus_{j=1}^{l} a_{ij} \otimes b_{jk} \\ &= \max_{1 \leq j \leq l} (a_{ij} + b_{jk}). \end{aligned} \tag{2-7}$$

Matrix multiplication is not commutative:

$$A \otimes B \neq B \otimes A. \tag{2-8}$$

Define the element-wise max-plus algebraic multiplication of two equally sized matrices $A \in \mathbb{R}_{\max}^{n \times m}$ and $B \in \mathbb{R}_{\max}^{n \times m}$ through:

$$[A \odot B]_{ij} = a_{ij} \otimes b_{ij} = a_{ij} + b_{ij} \tag{2-9}$$

Let $\mathcal{E}(n, m)$ be the $n \times m$ matrix with all elements equal to ε, then the following holds for all $n \times m$ matrices A:

$$A \oplus \mathcal{E}(n, m) = \mathcal{E}(n, m) \oplus A = A \tag{2-10}$$
$$A \otimes \mathcal{E}(m, k) = \mathcal{E}(n, k) \quad \text{and} \quad \mathcal{E}(k, n) \otimes A = \mathcal{E}(k, m). \tag{2-11}$$

Define the square matrix E as

$$[E]_{ij} = \begin{cases} e & \text{for } i = j \\ \varepsilon & \text{otherwise.} \end{cases} \tag{2-12}$$

then the following holds for square matrices $A \in \mathbb{R}_{\max}^{n \times n}$:

$$A \otimes E = E \otimes A = A. \tag{2-13}$$

Note that matrix E is the max-plus algebraic identity matrix and $\mathcal{E}$ represents the max-plus algebraic zero matrix.
The next section presents a max-plus algebraic model for railway networks, based on the theory presented above.

2-3 Railway Network System Description

As introduced in Section 2-1, the railway network operates according to various constraints, like minimum headway time and running time constraints. This section will present a max-plus model which can be used to calculate all departure and arrival times of all trains in the network, subject to those constraints. First, all relevant operational constraints are formalized algebraically in Section 2-3-1 yielding a max-plus system description. Section 2-3-2 structures this system description for the nominal case (i.e. disturbance-free) and Section 2-3-3 provides a notation for the system description in the perturbed case.

2-3-1 Max-Plus System Description

As introduced in Chapter 1, the railway network operates according to a periodic timetable. Events are thus scheduled to repeat every T minutes. Let k denote a cycle counter, such that if all events have occurred for the $k-th$ time, cycle k is complete. Each train has train runs associated to the individual tracks in the network. Each train run i is characterized by a departure and an arrival event. A train run i is from now on referred to as simply by train i, such that we can speak of "*the departure of train* i" and "*the arrival of train* i". Trains i and j in the model may therefore be associated to the same physical train!
The k-th departure time of train i is denoted by $d_i(k)$ and the k-th arrival time of train i is denoted by $a_i(k)$. The following four operational constraints apply to $d_i(k)$ and $a_i(k)$.

Timetable constraints Each train is scheduled to depart and arrive at predetermined times, as provided by the timetable:

$$d_i(k) \geq r_i^d(k) \tag{2-14}$$
$$a_i(k) \geq r_i^a(k), \tag{2-15}$$

where $r_i^d(k)$ and $r_i^a(k)$ denote the scheduled departure and arrival time of train i, respectively.

Running constraints The arrival time of train i depends on its departure time from the previous track and the running time on that track.

$$a_i(k) \geq \theta_i^{\text{run}} + d_i(k), \tag{2-16}$$

where θ_i^{run} denotes the running time of the train associated to train i.

Dwell constraints The departure time of train i depends on its arrival time at a station or junction and the dwell time at that station or junction:

$$d_i(k) \geq \theta_{ij}^{\text{dwell}} + a_j(k - \mu_{ij}), \tag{2-17}$$

where $\mu_{ij} = 0$ for events from the same cycle or $\mu_{ij} = 1$ if the arrival event is from the previous cycle. Note that events i and j are related to the same train, of which the dwell time is given by $\theta_{ij}^{\text{dwell}}$. Dwell constraints are therefore also called *continuity* constraints, as they link the movement of a train to different event numbers.

Headway constraints Trains have to be separated by a headway time τ_f when traveling in equal direction. Suppose that train j precedes train i, then

$$d_i(k) \geq \tau_f + d_j(k - \mu_{ij}) \tag{2-18}$$
$$a_i(k) \geq \tau_f + a_j(k - \mu_{ij}), \tag{2-19}$$

where $\mu_{ij} = 0$ for events from the same cycle or $\mu_{ij} = 1$ if d_j or a_j is from the previous cycle. Trains i and j are different physical trains!
When traveling in opposite directions (when meeting at a junction as in Figure 2-2):

$$d_i(k) \geq \tau_w + a_j(k - \mu_{ij}) \tag{2-20}$$

where $\mu_{ij} = 0$ for events from the same cycle or $\mu_{ij} = 1$ if the arrival event is from the previous cycle. Trains i and j use a virtual track. Again, trains i and j are different physical trains! Headway relations separate the occurrence of two events, associated to two different trains, in time and thus represent the order of trains.

Suppose that n is the total number of train runs and $1 \leq \ell \leq n$. The departure time of train i then follows from:

$$d_i(k) = \max_{\ell}(r_i^d(k), \theta_{i\ell}^{\text{dwell}} + a_\ell(k - \mu_{i\ell}), \tau_f + a_\ell(k - \mu_{i\ell}), \tau_h + d_\ell(k - \mu_{i\ell})). \tag{2-21}$$

The arrival time of train i follows from:

$$a_i(k) = \max_{\ell}(r_i^a(k), \theta_i^{\text{run}} + d_i(k), \tau_f + a_\ell(k - \mu_{i\ell})). \tag{2-22}$$

Suppose that $x_i(k)$ denotes an event time, which can either be a departure or an arrival, then this time is given by Eq. (2-23).

$$x_i(k) = \max_{j}(r_i(k), \theta_i^{\text{run}} + x_i(k), \theta_{ij}^{\text{dwell}} + x_j(k - \mu_{ij}), \tau_f + x_j(k - \mu_{ij}), \tau_h + x_j(k - \mu_{ij})), \tag{2-23}$$

where $1 \leq j \leq 2n$ as there are n departure and n arrival events.
Denote an activity time, like a running or dwell time, from event j to i in one cycle k ($\mu_{ij} = 0$) by a_{ij} and an activity time from event j in cycle $k - 1$ to event i in cycle k ($\mu_{ij} = 1$) by b_{ij} and by using Eq. (2-1) and Eq. (2-7), then Eq. (2-23) becomes:

$$x_i(k) = (\bigoplus_{j=1}^{n} a_{ij} \otimes x_j(k)) \oplus (\bigoplus_{j=1}^{n} b_{ij} \otimes x_j(k-1)) \oplus r_i(k), \tag{2-24}$$

If there is an activity from event j to i and they are in the same cycle k, then a_{ij} will have a non-ε value. If they are not in the same cycle, b_{ij} is non-ε.
When $x(k) \in \mathbb{R}_{\max}^{m\times 1}$ is a vector of all event times (departures and arrivals), a_{ij} and b_{ij} are matrix entries of respectively $A \in \mathbb{R}_{\max}^{n\times m}$ and $B \in \mathbb{R}_{\max}^{n\times m}$, and $r(k) \in \mathbb{R}_{\max}^{m\times 1}$ is a vector with scheduled event times. By applying Eq. (2-5) and Eq. (2-7), Eq. (2-24) becomes:

$$x_i(k) = [A \otimes x(k)]_i \oplus [B \otimes x(k-1)]_i \oplus r_i(k) \tag{2-25}$$

For the vector $x(k)$ then holds:

$$x(k) = A \otimes x(k) \oplus B \otimes x(k-1) \oplus r(k) \tag{2-26}$$

2-3-2 Nominal System Description

The previous section presented a max-plus system description for a railway network. This section aims at providing a structuring in this system description by partitioning the state vector $x(k)$ and grouping the system matrices A and B with respect to the various activity times. This provides a convenient notation for analysis and implementation. The partitioning of the system is chosen such that departure and arrival times are grouped in the event time vector $x(k)$ as shown in Eq. (2-27), where $d(k)$ and $a(k)$ denote the departure and arrival times of trains in cycle k, respectively. The vector containing schedule times is partitioned likewise (Eq. (2-27)), with $r^d(k)$ and $r^a(k)$ denoting scheduled departure and arrival times for cycle k, respectively. The same partitioning holds for $x(k-1)$.

$$x(k) = \left[\begin{array}{c} d(k) \\ \hline a(k) \end{array}\right], \quad r(k) = \left[\begin{array}{c} r^d(k) \\ \hline r^a(k) \end{array}\right] \tag{2-27}$$

The matrices are divided in four blocks, each representing a different type of constraint (as presented in Section 2-3-1) related to departures and arrivals. Eq. (2-28) shows this partitioning.

$$A = \left[\begin{array}{c|c} A_1 & A_2 \oplus A_3 \\ \hline A_4 & A_1 \end{array}\right], \quad B = \left[\begin{array}{c|c} B_1 & B_2 \oplus B_3 \\ \hline B_4 & B_1 \end{array}\right] \tag{2-28}$$

In Eq. (2-28), matrices A_1 and B_1 contain headway times for trains traveling in the same direction, whereas matrices A_2 and B_2 contain headway times for trains traveling in opposite directions. The former headway times are denoted by τ_f and the latter by τ_w. The entries of matrices A_1, B_1, A_2 and B_2 are defined as:

$$[A_1]_{ij} = \begin{cases} \tau_f & \text{if event } j \text{ precedes event } i \\ \varepsilon & \text{otherwise} \end{cases} \tag{2-29}$$

$$[B_1]_{ij} = \begin{cases} \tau_f & \text{if event } j \text{ from } k-1 \text{ precedes event } i \text{ from } k \\ \varepsilon & \text{otherwise} \end{cases} \tag{2-30}$$

$$[A_2]_{ij} = \begin{cases} \tau_w & \text{if event } j \text{ precedes event } i \text{ in opposite direction} \\ \varepsilon & \text{otherwise} \end{cases} \tag{2-31}$$

$$[B_2]_{ij} = \begin{cases} \tau_w & \text{if event } j \text{ from } k-1 \text{ precedes event } i \text{ from } k \text{ in opposite direction} \\ \varepsilon & \text{otherwise} \end{cases} \tag{2-32}$$

The dwell times are contained in matrices A_3 and B_3. The entries of matrices A_3 and B_3 are given in Eq. (2-35) and Eq. (2-34). Matrices A_4 and B_4 contain the running times. However,

since the convention is followed that a train departing in cycle k also has its arrival time defined in cycle k (see Section 2-3-1), all entries of matrix B_4 are equal to ε. Throughout this thesis, B_4 will therefore be denoted by $\mathcal{E}$. Let the vector containing all running times be denoted by Θ^{run} and the vector of all dwell times by Θ^{dwell}. These vectors are stacked to obtain the parameter vector Θ, as shown in Eq. (2-33).

$$\Theta = \left[\begin{array}{c} \Theta^{\text{run}} \\ \hline \Theta^{\text{dwell}} \end{array}\right] \tag{2-33}$$

Let $\gamma(i)$ denote the index in Θ^{dwell} of the dwell time related to train i. The entries of matrices B_3 and A_3 are given as:

$$[B_3]_{ij} = \begin{cases} \Theta^{\text{dwell}}_{\gamma(i)} & \text{if train } j \text{ from cycle } k-1 \text{ proceeds as train } i \text{ in cycle } k \\ \varepsilon & \text{otherwise,} \end{cases} \tag{2-34}$$

and

$$[A_3]_{ij} = \begin{cases} \Theta^{\text{dwell}}_{\gamma(i+n)} & \text{if train } j \text{ proceeds as train } i \text{ in the same cycle} \\ \varepsilon & \text{otherwise.} \end{cases} \tag{2-35}$$

Matrix A_4 is a diagonal matrix with on its diagonal the running times:

$$A_4 = \begin{bmatrix} \Theta^{\text{run}}_1 & \varepsilon & \dots & \varepsilon \\ \varepsilon & \Theta^{\text{run}}_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \varepsilon \\ \varepsilon & \dots & \varepsilon & \Theta^{\text{run}}_n \end{bmatrix} \tag{2-36}$$

where n is the number of trains in one cycle.

Since matrices A_3, A_4 and B_3 depend on Θ, Eq. (2-28) is rewritten into

$$A = \left[\begin{array}{c|c} A_1 & A_2 \oplus A_3(\Theta) \\ \hline A_4(\Theta) & A_1 \end{array}\right], \quad B = \left[\begin{array}{c|c} B_1 & B_2 \oplus B_3(\Theta) \\ \hline B_4 & B_1 \end{array}\right]. \tag{2-37}$$

Recall that the railway network model contains only those points where order swaps can occur, viz. stations and junctions, and that it consists of tracks connecting those points. Let $y(k)$ denote $x(k)$ ordered with respect to individual tracks and let $T \in \mathbb{R}^{m\times m}_{\max}$, such that $T^T \otimes T = E$, denote the transformation matrix needed to obtain $x(k)$ from $y(k)$, which is given in formula by:

$$x(k) = T \otimes y(k) = T \otimes \begin{bmatrix} x^1(k) \\ x^2(k) \\ \vdots \\ x^{n_t}(k) \end{bmatrix}, \tag{2-38}$$

where x^p denotes all event times associated to track p, for $1 \leq p \leq n_t$.
Inserting Eq. (2-38) into Eq. (2-26) yields:

$$\begin{aligned} y(k) &= T^T \otimes A \otimes T \otimes y(k) \oplus T^T \otimes B \otimes T \otimes y(k-1) \oplus r_p(k) \\ &= G \otimes y(k) \oplus H \otimes y(k-1) \oplus s(k) \end{aligned} \tag{2-39}$$

Like matrices A and B in Eq. (2-28), matrices G and H can be partitioned likewise:

$$G = \left[\begin{array}{c|c} G_1 & G_2 \oplus G_3(\Theta) \\ \hline G_4(\Theta) & G_1 \end{array}\right], \quad H = \left[\begin{array}{c|c} H_1 & H_2 \oplus H_3(\Theta) \\ \hline \mathcal{E} & H_1 \end{array}\right] \tag{2-40}$$

Let the transformation matrix T also be partitioned as

$$T = \left[\begin{array}{c|c} T_1 & T_2 \\ \hline T_3 & T_1 \end{array}\right], \quad \text{and} \quad T^T = \left[\begin{array}{c|c} T_1^T & T_3^T \\ \hline T_2^T & T_1^T \end{array}\right], \tag{2-41}$$

then the track-ordered headway matrices G_1, G_2, H_1 and H_2 are obtained through Eq. (2-42) and Eq. (2-43).

$$G_1 = \begin{bmatrix} G_1^1 & \mathcal{E} & \cdots & \mathcal{E} \\ \mathcal{E} & G_1^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathcal{E} \\ \mathcal{E} & \cdots & \mathcal{E} & G_1^{n_t} \end{bmatrix} = T_1^T \otimes A_1 \otimes T_1, \quad G_2 = \begin{bmatrix} G_2^1 & \mathcal{E} & \cdots & \mathcal{E} \\ \mathcal{E} & G_2^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathcal{E} \\ \mathcal{E} & \cdots & \mathcal{E} & G_2^{n_t} \end{bmatrix} = T_1^T \otimes A_2 \otimes T_3 \tag{2-42}$$

$$H_1 = \begin{bmatrix} H_1^1 & \mathcal{E} & \cdots & \mathcal{E} \\ \mathcal{E} & H_1^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathcal{E} \\ \mathcal{E} & \cdots & \mathcal{E} & H_1^{n_t} \end{bmatrix} = T_1^T \otimes B_1 \otimes T_1, \quad H_2 = \begin{bmatrix} H_2^1 & \mathcal{E} & \cdots & \mathcal{E} \\ \mathcal{E} & H_2^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathcal{E} \\ \mathcal{E} & \cdots & \mathcal{E} & H_2^{n_t} \end{bmatrix} = T_1^T \otimes B_2 \otimes T_3 \tag{2-43}$$

2-3-3 Perturbed System Description

The system description presented in Eq. (2-39) reflects nominal operation and (for a feasible timetable) $y(k)$ will be equal to $s(k)$. It is furthermore assumed that $y(k-1)$ is fully known. During operation, however, not all parameters are known beforehand. For example, the running time of a train will not be known until its arrival. This running time will be nominal in a disturbance-free case, but will deviate (i.e. will be bigger) in a disturbed case. Therefore, the parameter set Θ may differ from cycle to cycle and an appropriate notation will thus be $\Theta(k)$. Eq. (2-39) can then be written as

$$y(k) = G(\Theta(k)) \otimes y(k) \oplus H(\Theta(k)) \otimes y(k-1) \oplus s(k) \tag{2-44}$$

As at a certain point in time, part of $\Theta(k)$ is known and the remainder will be an estimation of the actual parameter values. Therefore, $\Theta(k)$ may also change with time and at a certain time instant, a prediction of $\Theta(k)$ is available and an appropriate notation would be $\Theta(k \mid t)$. Eq. (2-44) then becomes:

$$y(k \mid t) = G(\Theta(k \mid t)) \otimes y(k \mid t) \oplus H(\Theta(k \mid t)) \otimes y(k-1) \oplus s(k) \tag{2-45}$$

State vector $y(k)$ then also contains predicted event times susceptible to change over time, hence the notation $y(k \mid t)$. Vector $y(k-1)$ was assumed to be fully known and therefore does not change in time whilst the system is in cycle k. Note that even though $y(k-1)$ is known, dwell activities relating events from $k-1$ to k can still be perturbed.
Eq. (2-28) then becomes:

$$G(\Theta(k \mid t)) = \left[\begin{array}{c|c} G_1 & G_2 \oplus G_3(\Theta(k \mid t)) \\ \hline G_4(\Theta(k \mid t)) & G_1 \end{array}\right] \tag{2-46}$$

$$H(\Theta(k \mid t)) = \left[\begin{array}{c|c} H_1 & H_2 \oplus H_3(\Theta(k \mid t)) \\ \hline \mathcal{E} & H_1 \end{array}\right] \tag{2-47}$$

2-4 Conclusions

In this chapter, a max-plus linear model of a railway network was presented. This model is structured with respect to the headway, running and dwell times. The running and dwell times are stacked in the parameter vector $\Theta(k \mid t)$. Since this parameter vector is susceptible to perturbations, part of this vector is known and the remainder is estimated. The next chapter presents a controllable system description, in which the headway time matrices can be manipulated by a control variable.

Chapter 3

Control

This chapter presents how order swaps are implemented [22] into the max-plus system description presented in the previous chapter. First, an example is presented to get insight into how order swaps are implemented. Then, the structured perturbed max-plus model is extended to also be structured in the input vector in Section 3-2. This model is used as basis for a prediction model, which is developed in Section 3-3. Section 3-4 presents the MILP formulation [22] of finding the optimal inputs and event times. The receding horizon control framework as used for the control of railway networks is presented in Section 3-5.

3-1 Order Swap Example

As mentioned in Section 2-3-1, headway relations represent the order of trains. This section shows an example in which the headway relations between two trains is modified to implement an order swap. Consider the simple network consisting of three stations A, B and C connected by two tracks, as depicted in Figure 3-1.

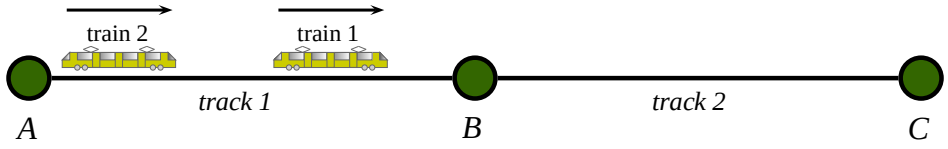

Figure 3-1: Example network in which train 1 acquires a delay at station B.

Suppose that during each cycle, two train lines operate on the network. Line 1 has planned stops at all three stations, whereas line 2 only stops at stations A and C. Train a operates on line 1 and train b operates on line 2. Assume that at each station there is the possibility for trains to overtake each other. Furthermore assume that all events of the previous cycle have occurred and were punctual (i.e. all trains were on time). In other words, $y(k-1)$ is fully known and equal to $s(k-1)$. Suppose that train a precedes train b on both tracks in the nominal case, but that due to a disturbance the departure of train a from station B is

delayed. If no action would be taken, train b would have to wait for train a to leave station B and would as a result also be delayed. The departure of train b should thus be given priority above that of train a. For this example it is sufficient to only consider the departure events and matrix G_1, since G_1 also relates the arrival events and there are no trains running in opposite directions ($G_2 = \mathcal{E}$). Since two trains are using two tracks, there is a total of 4 train runs (with 4 departures and 4 arrivals) each cycle. Recall that a train run i is referred to as a train i. Define the vector $d(k)$ as follows:

$$d(k) = \begin{bmatrix} d_1(k) \\ d_2(k) \\ d_3(k) \\ d_4(k) \end{bmatrix} = \begin{bmatrix} \text{departure of train 1 from track 1} \\ \text{departure of train 2 from track 1} \\ \text{departure of train 3 from track 2} \\ \text{departure of train 4 from track 2} \end{bmatrix}. \tag{3-1}$$

where trains 1 and 3 are associated to train a and trains 2 and 4 are associated to train b. The departure events are related through G_1 as follows:

$$d(k) = G_1 \otimes d(k) = \begin{bmatrix} G_1^1 & \mathcal{E} \\ \mathcal{E} & G_1^2 \end{bmatrix} \otimes d(k). \tag{3-2}$$

Matrices G_1^1 and G_1^2 are given by:

$$G_1^1 = \begin{bmatrix} \varepsilon & \varepsilon \\ \tau_f & \varepsilon \end{bmatrix}, \quad G_1^2 = \begin{bmatrix} \varepsilon & \varepsilon \\ \tau_f & \varepsilon \end{bmatrix}. \tag{3-3}$$

The departure order of train 1 and train 2 from track 2 needs to be swapped. This can be achieved by swapping entries $[G_1^2]_{2,1}$ and $[G_1^2]_{1,2}$. In that way, the departure time of train 2 from track 2 now follows from the departure time of train 1 from track 2: $d_3(k) = d_4(k) \otimes \tau_f$. Matrix G_1^2 thus becomes:

$$G_1^2 = \begin{bmatrix} \varepsilon & \tau_f \\ \varepsilon & \varepsilon \end{bmatrix}. \tag{3-4}$$

In general, the procedure of swapping orders between events i and j is as follows. Suppose that in the nominal case, event j from cycle k precedes event i from cycle k, such that $[G_1]_{ij} = \tau_f$ and $[G_1]_{ji} = \varepsilon$ (for the same direction) or $[G_2]_{ij} = \tau_w$ and $[G_2]_{ji} = \varepsilon$ (for opposite directions). Then, an order swap is implemented by setting $[G_1]_{ij} = \varepsilon$ and $[G_1]_{ji} = \tau_f$ or $[G_2]_{ij} = \varepsilon$ and $[G_2]_{ji} = \tau_w$. If event j belongs to cycle $k-1$ and i to cycle k, the swap is applied to H_1 or H_2.

3-2 Controlled System Description

In the previous section, it was demonstrated that order swaps are realized through manipulation of the headway matrices. More specifically, by swapping entries of the track-order matrices G_1, G_2, H_1 and H_2. This section will present the mathematical formulation for realizing order swaps, in stead of doing this by inspection as was done in the example of the

previous section.
First, the matrices G_1, G_2, H_1 and H_2 are split as follows:

$$G_1^p = M^p \odot D^p \odot \tau_f \tag{3-5}$$
$$G_2^p = M^p \odot \bar{D}^p \odot \tau_w \tag{3-6}$$
$$H_1^p = N^p \odot D^p \odot \tau_f \tag{3-7}$$
$$H_2^p = N^p \odot \bar{D}^p \odot \tau_w. \tag{3-8}$$

Matrix M^p defines event orders on track p, matrices D^p and $\bar{D}^p$ are direction matrices for track p and matrix N^p defines event orders on track p for two events from cycle $k-1$ and k. Define the very large negative number $\beta << 0$. The entries of matrices M^p, N^p, D^p and $\bar{D}^p$ are given by Eq. (3-9) - Eq. (3-12).

$$[M^p]_{ij} = \begin{cases} 0 & \text{if event } j \text{ precedes event } i \text{ on track } p \\ \beta & \text{otherwise} \end{cases} \tag{3-9}$$

$$[N^p]_{ij} = \begin{cases} 0 & \text{if event } j \text{ from cycle } k-1 \text{ precedes event } i \text{ from cycle } k \text{ on track } p \\ \beta & \text{otherwise} \end{cases} \tag{3-10}$$

$$[D^p]_{ij} = \begin{cases} 0 & \text{for trains traveling in equal direction on track } p \\ \beta & \text{for trains traveling in opposite direction on track } p \end{cases} \tag{3-11}$$

$$[\bar{D}^p]_{ij} = \begin{cases} 0 & \text{for trains traveling in opposite direction on track } p \\ \beta & \text{for trains traveling in equal direction on track } p \end{cases} \tag{3-12}$$

Order swaps on track p can now be implemented by manipulating the entries of M^p, without having to take the direction and the associated different headway time (τ_f or τ_w) into consideration.
Define $u_M^p \in \{0,1\}^{n_u}$, with $n_u = n_p(n_p-1)/2$, where n_p equals the number of trains on track p during one cycle, then in cycle k for each track matrix M^p can be manipulated through use of Eq. (3-13). This permutation of matrix M^p is referred to as *mode switching*, as for each choice of u_M^p the system description changes or switches to another operational mode. The introduction of β to represent ε in the controlled system description ensures that the notation using multiplication from conventional algebra, as used in Eq. (3-13), is correct.

$$M^p(u_M^p(k)) = M_0^p \odot \bigodot_{l=1}^{n_p(n_p-1)/2} M_l^p \cdot [u_M^p(k)]_l, \tag{3-13}$$

where matrix M_0^p represents order matrix M^p in the nominal case, matrix M_l^p represents a permutation option and the binary variable $[u_M^p(k)]_l$ is the input associated to that specific permutation. The input $[u_M^p(k)]_l$ thus activates the permutation M_l^p.
In the same way, order swaps between events from subsequent cycles can be implemented by permutation of matrix N^p:

$$N^p(u_N^p(k)) = N_0^p \odot \sum_{l=1}^{n_p(n_p-1)/2} N_l^p \cdot [u_N^p(k)]_l, \tag{3-14}$$

where matrix N_0^p represents order matrix N^p in the nominal case, matrix N_l^p represents a permutation option and the binary variable $[u_N^p(k)]_l$ is the input associated to that specific permutation.
Suppose that mode l is associated to events i and j, where j precedes i on track p in the nominal case, then the entries of M_l^p and N_l^p are defined as

$$[M_l^p]_{qr} = \begin{cases} \beta & \text{for } q = i \text{ and } r = j \\ -\beta & \text{for } q = j \text{ and } r = i \\ 0 & \text{otherwise,} \end{cases} \tag{3-15}$$

and

$$[N_l^p]_{qr} = \begin{cases} \beta & \text{for } q = i \text{ and } r = j \\ -\beta & \text{for } q = j \text{ and } r = i \\ 0 & \text{otherwise.} \end{cases} \tag{3-16}$$

In the permutable system description, matrices G_1 and G_2 depend on $u_M(k)$, where $u_M(k) = [u_M^1(k)\ u_M^2(k)\ \cdots u_M^{n_t}(k)]^T$. Eq. (2-46) can thus be rewritten into

$$G(u_M(k), \Theta(k \mid t)) = \begin{bmatrix} G_1(u_M(k)) & G(u_M(k)) \oplus G_3(\Theta(k \mid t)) \\ G_4(\Theta(k \mid t)) & G_1(u_M(k)) \end{bmatrix}. \tag{3-17}$$

Matrices H_1 and H_2 depend on $u_N(k)$, where $u_N(k) = [u_N^1(k)\ u_N^2(k)\ \cdots u_N^{n_t}(k)]^T$. Equation 2-47 can thus be rewritten into

$$H(u_N(k-1), \Theta(k \mid t)) = \begin{bmatrix} H_1(u_N(k-1)) & H_2(u_N(k-1)) \oplus H_3(\Theta(k \mid t)) \\ \mathrm{B} & H_1(u_N(k-1)) \end{bmatrix}. \tag{3-18}$$

where B is a matrix with its entries equal to β. When swapping orders of trains it may occur that an event needs to be scheduled after an event scheduled to occur in the next cycle (i.e. cycle $k+1$). Therefore, in the controlled case, $y(k)$ may also depend on $y(k+1)$. First, Eq. (2-45) is augmented with the term $I(k) \otimes y(k+1 \mid t)$:

$$y(k \mid t) = G(\Theta(k \mid t)) \otimes y(k \mid t) \oplus H(\Theta(k \mid t)) \otimes y(k-1) \oplus I(k) \otimes y(k+1 \mid t) \oplus s(k). \tag{3-19}$$

In the nominal case, $I(k) = \mathcal{E}$. Like matrices $G(\Theta(k \mid t))$ and $H(\Theta(k \mid t))$, matrix $I(k)$ is partitioned as follows:

$$I((k)) = \begin{bmatrix} I_1(k) & I_2(k) \\ \mathrm{B} & I_1(k) \end{bmatrix}, \tag{3-20}$$

where

$$[I_1(k)]_{ij} = \begin{cases} \tau_f & \text{if event } j \text{ precedes event } i \\ \beta & \text{otherwise} \end{cases} \tag{3-21}$$

and

$$[I_2(k)]_{ij} = \begin{cases} \tau_w & \text{if event } j \text{ precedes event } i \text{ in opposite direction} \\ \beta & \text{otherwise} \end{cases} \tag{3-22}$$

The entries of matrices I_1 and I_2 follow from:

$$I_1^p = O^p \odot D^p \odot \tau_f \tag{3-23}$$
$$I_2^p = O^p \odot \bar{D}^p \odot \tau_w, \tag{3-24}$$

where the entries of O^p is defined as:

$$[O^p]_{ij} = \begin{cases} 0 & \text{if event } j \text{ from cycle } k+1 \text{ precedes event } i \text{ from cycle } k \text{ on track } p \\ \beta & \text{otherwise} \end{cases} \tag{3-25}$$

Define $u_O^p \in \{0,1\}^{n_u}$, with $n_u = n_p(n_p-1)/2$, where n_p equals the number of trains on track p during one cycle, then in cycle k for each track matrix O^p can be manipulated through

$$O^p(u_O^p(k)) = O_{p,0} \odot \sum_{l=1}^{n_p(n_p-1)/2} O_l^p \cdot [u_O^p(k)]_l \tag{3-26}$$

where O_0^p represents order matrix O^p in the nominal case, O_l^p represents a permutation option and $[u_O^p(k)]_l$ is the input associated to that specific permutation.
Suppose that mode l is associated to events i and j, where j precedes i on track p in the nominal case, then the entries of O_l^p are defined as

$$[O_l^p]_{qr} = \begin{cases} \beta & \text{for } q = i \text{ and } r = j \\ -\beta & \text{for } q = j \text{ and } r = i \\ 0 & \text{otherwise.} \end{cases} \tag{3-27}$$

Matrix $I(k)$ thus depends on $u_O(k)$, therefore we will use the notation $I(u_O(k))$ and Eq. (3-20) becomes:

$$I(u_O(k)) = \begin{bmatrix} I_1(u_O(k)) & I_2(u_O(k)) \\ \mathrm{B} & I_1(u_O(k)) \end{bmatrix}, \tag{3-28}$$

Eq. (3-19) is rewritten into:

$$\begin{aligned} y(k \mid t) = G(\Theta(k \mid t)) \otimes y(k \mid t) \oplus H(\Theta(k \mid t)) \otimes y(k-1) \\ \oplus I(u_O(k)) \otimes y(k+1 \mid t) \oplus s(k). \end{aligned} \tag{3-29}$$

When new measurements become available for events in cycle k at time t, the earlier computed input sequence may not be optimal any longer and needs to be recomputed. Therefore, the input vectors may also change within one cycle at various time instants and Eq. (3-29) becomes:

$$\begin{aligned} y(k \mid t) = G\big(u_M(k,t), \Theta(k \mid t)\big) \otimes y(k \mid t) \oplus H\big(u_N(k-1,t), \Theta(k \mid t)\big) \otimes y(k-1) \\ \oplus I\big(u_O(k,t)\big) \otimes y(k+1 \mid t) \oplus s(k). \end{aligned} \tag{3-30}$$

The next section extends Eq. (3-30) over an horizon of future cycles, such that consequences for future cycles of actions taken now can be assessed during the current cycle.

3-3 Prediction Model

This section presents a description of the prediction model based on the model description of Section 3-2. This prediction model predicts event times for cycles k to $k + N_p$, where N_p denotes the prediction horizon. The event times for each cycle within the prediction horizon are given by Equation 3-31.

$$\begin{aligned} y(k+j \mid t) = G(u_M(k+j,t), \Theta(k+j \mid t)) \otimes y(k+j \mid t) \ \oplus \dots \\ H(u_N(k+j-1,t), \Theta(k+j \mid t)) \otimes y(k+j-1) \ \oplus \dots \\ I(u_O(k+j,t)) \otimes y(k+j+1 \mid t) \oplus s(k+j) \\ \text{for } j = 0, \dots, N_p \end{aligned} \tag{3-31}$$

Define $\tilde{y}(k \mid t)$ as the vector containing event times over the prediction horizon as predicted in cycle k at time instant t. Its structure is given in Equation 3-32, where $\tilde{s}(k)$ is represented likewise. The same is done for the input and parameter vector, yielding $\tilde{u}(k,t)$ and $\tilde{\Theta}(k \mid t)$ respectively. Their structure is shown in Equation 3-33

$$\tilde{y}(k \mid t) = \begin{bmatrix} d(k \mid t) \\ d(k+1 \mid t) \\ \vdots \\ d(k+N_p \mid t) \\ \hline a(k \mid t) \\ a(k+1 \mid t) \\ \vdots \\ a(k+N_p \mid t) \end{bmatrix} = \begin{bmatrix} \tilde{d}(k \mid t) \\ \hline \tilde{a}(k \mid t) \end{bmatrix}, \quad \tilde{s}(k) = \begin{bmatrix} s_d(k) \\ s_d(k+1) \\ \vdots \\ s_d(k+N_p) \\ \hline s_a(k) \\ s_a(k+1) \\ \vdots \\ s_a(k+N_p) \end{bmatrix} = \begin{bmatrix} \tilde{s}_d(k) \\ \hline \tilde{s}_a(k) \end{bmatrix} \tag{3-32}$$

$$\tilde{u}(k,t) = \begin{bmatrix} u_M(k,t) \\ u_M(k+1,t) \\ \vdots \\ u_M(k+N_p,t) \\ \hline u_N(k) \\ u_N(k+1,t) \\ \vdots \\ u_N(k+N_p-1,t) \\ \hline u_O(k,t) \\ u_O(k+1,t) \\ \vdots \\ u_O(k+N_p,t) \end{bmatrix} = \begin{bmatrix} \tilde{u}_M(k,t) \\ \hline \tilde{u}_N(k,t) \\ \hline \tilde{u}_O(k,t) \end{bmatrix}, \quad \tilde{\Theta}(k \mid t) = \begin{bmatrix} \Theta^{\text{run}}(k \mid t) \\ \Theta^{\text{run}}(k+1 \mid t) \\ \vdots \\ \Theta^{\text{run}}(k+N_p \mid t) \\ \hline \Theta^{\text{dwell}}(k \mid t) \\ \Theta^{\text{dwell}}(k+1 \mid t) \\ \vdots \\ \Theta^{\text{dwell}}(k+N_p \mid t) \end{bmatrix} = \begin{bmatrix} \tilde{\Theta}^{\text{run}}(k \mid t) \\ \hline \tilde{\Theta}^{\text{dwell}}(k \mid t) \end{bmatrix} \tag{3-33}$$

The system generating $\tilde{y}(k \mid t)$, using Equation 3-33 is given by:

$$\tilde{y}(k \mid t) = \mathcal{G}\big(\tilde{u}(k,t), \tilde{\Theta}(k \mid t)\big) \otimes \tilde{y}(k \mid t) \oplus \mathcal{H}\big(u_N(k-1), \tilde{\Theta}(k \mid t)\big) \otimes y(k-1) \oplus \tilde{s}(k) \tag{3-34}$$

with

$$\mathcal{G}(\tilde{u}(k,t), \tilde{\Theta}(k \mid t)) = \left[\begin{array}{c|c} \mathcal{G}_1(\tilde{u}(k,t)) & \mathcal{G}_2(\tilde{u}(k,t)) \oplus \mathcal{G}_3(\tilde{\Theta}(k \mid t)) \\ \hline \mathcal{G}_4(\tilde{\Theta}(k \mid t)) & \mathcal{G}_1(\tilde{u}(k,t)) \end{array}\right] \tag{3-35}$$

and

$$\mathcal{H}\big(u_N(k-1), \tilde{\Theta}(k \mid t)\big) = \left[\begin{array}{c|c} \mathcal{H}_1(u_N(k-1)) & \mathcal{H}_2(u_N(k-1)) \oplus \mathcal{H}_3(\tilde{\Theta}(k \mid t)) \\ \hline \text{B} & \mathcal{H}_1(u_N(k-1)) \end{array}\right] \tag{3-36}$$

(3-37)

where

$$\mathcal{G}_1^p = \mathcal{M}^p(\tilde{u}^p(k)) \odot \mathcal{D}^p \odot \tau_f \tag{3-38}$$

$$\mathcal{G}_2^p = \mathcal{M}^p(\tilde{u}^p(k)) \odot \bar{\mathcal{D}}^p \odot \tau_w. \tag{3-39}$$

The entries of matrices $\mathcal{M}^p$, $\mathcal{D}^p$ and $\bar{\mathcal{D}}^p$ are defined as those of M^p, D^p and $\bar{D}^p$ (Eq. (3-9), Eq. (3-11) and Eq. (3-12), respectively).
Matrix $\mathcal{M}^p(\tilde{u}_p(k))$ depends on $\tilde{u}_p(k)$ in the following way:

$$\mathcal{M}^p(\tilde{u}^p(k)) = \mathcal{M}_0^p \odot \sum_{i=1}^{\tilde{n}_p(\tilde{n}_p-1)/2} \mathcal{M}_i^p[\tilde{u}^p(k)]_i, \tag{3-40}$$

where $\mathcal{M}_0^p$ represents the nominal situation on track p during the prediction horizon, $[\tilde{u}^p(k)]_i$ is the binary control variable associated to permutation option $\mathcal{M}_i^p$ on track p, and $\tilde{n}_p$ is the number of trains using track p during the prediction horizon.

Matrices $\mathcal{H}_1(u_N(k-1))$ and $\mathcal{H}_2(u_N(k-1))$ are defined as

$$\mathcal{H}_1(u_N(k-1)) = \begin{bmatrix} H_1(u_N(k-1)) \\ \mathrm{B} \\ \vdots \\ \mathrm{B} \end{bmatrix}, \quad \mathcal{H}_2(u_N(k-1)) = \begin{bmatrix} H_2(u_N(k-1)) \\ \mathrm{B} \\ \vdots \\ \mathrm{B} \end{bmatrix} \tag{3-41}$$

and $\mathcal{H}_3(\tilde{\Theta}(k \mid t))$ is defined through:

$$\mathcal{H}_3(\tilde{\Theta}(k \mid t)) = \begin{bmatrix} H_3(\tilde{\Theta}(k \mid t)) \\ \mathrm{B} \\ \vdots \\ \mathrm{B} \end{bmatrix}. \tag{3-42}$$

Matrices $\mathcal{G}_1(\tilde{u}(k,t))$, $\mathcal{G}_2(\tilde{u}(k,t))$, $\mathcal{G}_3(\tilde{\Theta}(k \mid t))$ and $\mathcal{G}_4(\tilde{\Theta}(k \mid t))$ are defined in Equations 3-44 - 3-47, where the arguments k and t are left out for presentational reasons only. Note from Eq. (3-44) that matrices $I_1(u_N(k))$ and $H_1(u_N(k))$ are coupled through u_N, since a permutation of $I_1(u_N(k))$ relating $y(k)$ to $y(k+1)$ in cycle k, will define permutations to $H_1(u_N(k))$ in cycle $k+1$. This implies that for the prediction model it does not make sense to differ between $u_N(k)$ and $u_O(k-1)$. The same holds for I_2 and H_2. Therefore, Equations 3-44 - 3-47 do not contain u_O and the vector $\tilde{u}(k,t)$ is redefined in Equation 3-43.
Matrix $I_1(u_N(N_p))$ is not included in the expression for $y(k+N_p)$ in Equation 3-44, since $y(k+N_p+1)$ falls outside the prediction horizon. The same holds for I_2.

$$\tilde{u}(k,t) = \left[\begin{array}{c} u_M(k,t) \\ u_M(k+1,t) \\ \vdots \\ u_M(k+N_p,t) \\ \hline u_N(k) \\ u_N(k+1,t) \\ \vdots \\ u_N(k+N_p-1,t) \end{array}\right] = \left[\begin{array}{c} \tilde{u}_M(k,t) \\ \hline \tilde{u}_N(k,t) \end{array}\right] \tag{3-43}$$

$$\mathcal{G}_1(\tilde{u}(k)) = \begin{bmatrix} G_1(u_M(0)) & I_1(u_N(0)) & \mathrm{B} & & \\ H_1(u_N(0)) & G_1(u_M(1)) & I_1(u_N(1)) & \ddots & \vdots \\ \ddots & \ddots & \ddots & \ddots & \mathrm{B} \\ & & H_1(u_N(_{N_p}-2)) & G_1(u_M(_{N_p}-1)) & I_1(u_N(N_p-1)) \\ \dots & \mathrm{B} & \mathrm{B} & H_1(u_N(_{N_p}-1)) & G_1(u_M(N_p)) \end{bmatrix} \tag{3-44}$$

$$\mathcal{G}_2(\tilde{u}(k)) = \begin{bmatrix} G_2(u_M(0)) & I_2(u_N(0)) & \mathrm{B} & & \\ H_2(u_N(0)) & G_2(u_M(1)) & I_2(u_N(1)) & \ddots & \vdots \\ \ddots & \ddots & \ddots & \ddots & \mathrm{B} \\ & & H_2(u_N(_{N_p}-2)) & G_2(u_M(_{N_p}-1)) & I_2(u_N(N_p-1)) \\ \dots & \mathrm{B} & \mathrm{B} & H_2(u_N(_{N_p}-1)) & G_2(u_M(N_p)) \end{bmatrix} \tag{3-45}$$

$$\mathcal{G}_3(\tilde{\Theta}(k)) = \begin{bmatrix} G_3(\Theta(0)) & \mathrm{B} & & & \\ H_3(\Theta(0)) & G_3(\Theta(1)) & \ddots & & \vdots \\ \ddots & \ddots & \ddots & & \\ & & H_3(\Theta(N_p-1)) & G_3(\Theta(N_p-1)) & \mathrm{B} \\ \dots & \mathrm{B} & \mathrm{B} & H_3(\Theta(N_p)) & G_3(\Theta(N_p)) \end{bmatrix} \tag{3-46}$$

$$\mathcal{G}_4(\tilde{\Theta}(k)) = \begin{bmatrix} G_4(\Theta(0)) & \mathrm{B} & & & \\ \mathrm{B} & G_4(\Theta(1)) & \ddots & & \\ \ddots & & \ddots & & \vdots \\ & & & G_4(\Theta(N_p-1)) & \mathrm{B} \\ & \dots & & \mathrm{B} & G_4(\Theta(N_p)) \end{bmatrix} \tag{3-47}$$

3-4 Solving the Control Problem

This section presents a linear programming formulation to solve the problem of finding optimal the optimal input and state vector. It has been shown in [22] that this problem yields a *Mixed Integer Linear Programming* problem. The optimization has to minimize some cost function $J(k)$, through the optimizers $\tilde{y}(k \mid t)$ and $\tilde{u}(k,t)$:

$$\min_{\tilde{y}(k|t),\tilde{u}(k,t)} J(k) \tag{3-48}$$

Objective function As the objective is to minimize the total delay over the whole prediction horizon, an obvious candidate cost function is:

$$J(k) = c_y^T \Big(\tilde{y}(k \mid t) - \tilde{s}(k) \Big), \tag{3-49}$$

where c_y^T is a vector containing constant weights. Since the constant term $-c_y^T \tilde{s}(k)$ does not affect the position of the optimum, it can be discarded from the cost function:

$$J(k) = c_y^T \tilde{y}(k \mid t), \tag{3-50}$$

When changing train order, it may occur that the non-prioritized train has to decelerate (to a full stop) and accelerate again. The time needed for this action contributes to the delay. Therefore, an order swap is also penalized in the optimization, albeit with a weight much smaller than given to the state vector. The cost function then becomes:

$$J(k) = c_y^T \tilde{y}(k \mid t) + c_u^T \tilde{u}(k, t). \tag{3-51}$$

Constraints Eq. (3-34) can be written as (through applying Eq. (2-5) and Eq. (2-7)):

$$\begin{aligned} \tilde{y}_i(k \mid t) = \max(\tilde{s}_i(k), &\max_j (\tilde{y}_j(k \mid t) + [\mathcal{G}(\tilde{u}(k,t), \tilde{\Theta}(k \mid t))]_{ij}), \\ &\max_h (y_h(k-1) + [\mathcal{H}(u_N(k-1), \tilde{\Theta}(k \mid t))]_{ih})), \end{aligned} \tag{3-52}$$

which can be rewritten into (where $(\cdot \mid t)$ and $(\cdot, t)$ are left out of the arguments for presen-

tational reasons only)

$$\tilde{d}_i(k) \geq \tilde{s}_i^d(k) \tag{3-53a}$$
$$\tilde{a}_i(k) \geq \tilde{s}_i^a(k) \tag{3-53b}$$
$$\tilde{d}_i(k) \geq \tilde{a}_j(k) + [\mathcal{G}_3(\tilde{\Theta}(k))]_{ij} \quad \forall j \tag{3-53c}$$
$$\tilde{a}_i(k) \geq \tilde{d}_j(k) + [\mathcal{G}_4(\tilde{\Theta}(k))]_{ij} \quad \forall j \tag{3-53d}$$
$$\tilde{d}_i(k) \geq a_h(k-1) + [\mathcal{H}_3(\tilde{\Theta}(k))]_{ih} \quad \forall h \tag{3-53e}$$
$$\tilde{d}_i(k) \geq \tilde{d}_j(k) + \tau_f + [\mathcal{D}]_{ij} + [\mathcal{M}_0]_{ij} + \sum_l^{n_{\tilde{u}}} [\mathcal{M}_l]_{ij} \cdot [\tilde{u}(k)]_l \quad \forall j \tag{3-53f}$$
$$\tilde{d}_i(k) \geq \tilde{d}_j(k) + \tau_w + [\bar{\mathcal{D}}]_{ij} + [\mathcal{M}_0]_{ij} + \sum_l^{n_{\tilde{u}}} [\mathcal{M}_l]_{ij} \cdot [\tilde{u}(k)]_l \quad \forall j \tag{3-53g}$$
$$\tilde{a}_i(k) \geq \tilde{a}_j(k) + \tau_f + [\mathcal{D}]_{ij} + [\mathcal{M}_0]_{ij} + \sum_l^{n_{\tilde{u}}} [\mathcal{M}_l]_{ij} \cdot [\tilde{u}(k)]_l \quad \forall j \tag{3-53h}$$
$$\tilde{d}_i(k) \geq d_h(k-1) + \tau_f + [D]_{ih} + [N_0]_{ih} + \sum_l^{n_{u_N}} [N_l]_{ih} \cdot [u_N(k-1)]_l \quad \forall h \tag{3-53i}$$
$$\tilde{d}_i(k) \geq d_h(k-1) + \tau_w + [\bar{D}]_{ih} + [N_0]_{ih} + \sum_l^{n_{u_N}} [N_l]_{ih} \cdot [u_N(k-1)]_l \quad \forall h \tag{3-53j}$$
$$\tilde{a}_i(k) \geq a_h(k-1) + \tau_f + [D]_{ih} + [N_0]_{ih} + \sum_l^{n_{u_N}} [N_l]_{ih} \cdot [u_N(k-1)]_l \quad \forall h, \tag{3-53k}$$

where $h = 1, 2, \ldots, n$, $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$ are the train numbers, $n_{\tilde{u}}$ is the number of permutations associated to matrix $\mathcal{M}$ and n_{u_N} is the number of permutations associated to matrix N. These constraints are linear in $\tilde{d}(k \mid t)$ and $\tilde{a}(k \mid t)$, and therefore in $\tilde{y}(k \mid t)$, as well as in $\tilde{u}(k, t)$. Eq. (3-53) can thus be written in the form:

$$A_c \begin{bmatrix} \tilde{y}(k \mid t) \\ \tilde{u}(k, t) \end{bmatrix} \geq b_c(k) \tag{3-54}$$

The linear objective function (Eq. (3-51)) subject to the linear constraints, with continuous values for $\tilde{y}(k \mid t)$ and binary values for $\tilde{u}(k, t)$, yields a Mixed Integer Linear Programme (MILP).

3-5 Receding Horizon Control

In conventional receding horizon control, an optimal sequence of future inputs is computed, of which the first input is implemented in the real system. Then, the prediction horizon shifts ahead one sample and the process restarts. Figure 3-2 illustrates this process.

At time step l some objective function J is minimized subject to constraints, by optimizing a future control sequence $\{u(l \mid l, \ldots, u(l + N_c - 1 \mid l)\}$. At this time step l only $u(l \mid l)$ is applied to the process under control. At the next time step $(l + 1)$, the horizon shifts ahead

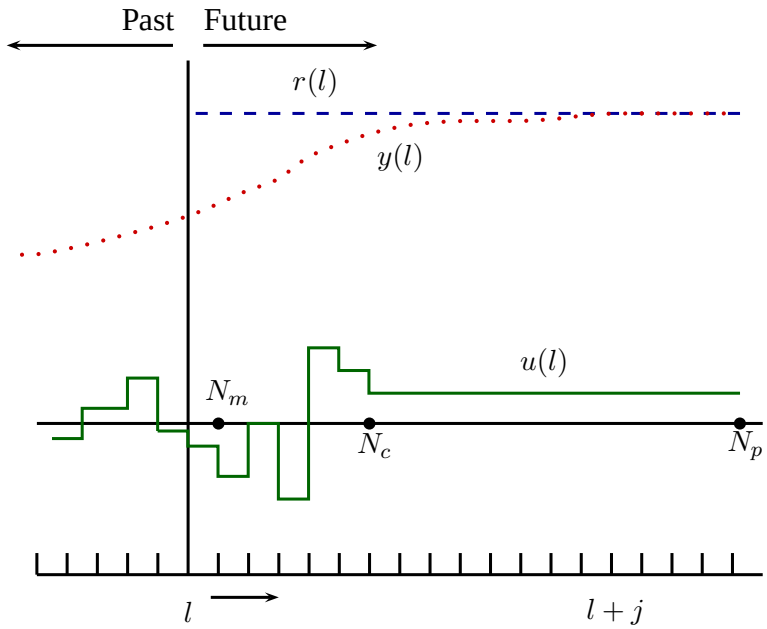

Figure 3-2: The principle of receding horizon control.

one sample and the optimization is repeated.
In Section 3-3 the prediction horizon was defined to be an integer number of cycles. During each cycle k new measurements become available at various time steps, which can either be actual event times, or estimated parameter values. During one cycle multiple optimizations are thus carried out using this updated information, yielding new optimal states and inputs for cycles k to $k + N_p$.

3-5-1 Event time constraints

Consider the situation in Figure 3-3. The graphs show the number of pending events in the prediction horizon plotted against time. The upper graph shows when the system is in cycle k and predicts event times up to and including cycle $k+1$ (as indicated by the dashed area). All events of cycle $k-1$ are known. Assume that cycle k will not be perturbed, such that no control is needed. As time progresses, more state information becomes available and the number of pending events thus decreases. At the point t_2, this number increases, since then all event times of cycle k are known and the horizon shifts ahead one cycle to $k' = k+1$ and includes $k'+1 = k+2$, which is depicted in the middle graph. This is consistent with the assumption that all event times of the initial cycle $k-1$ are known, as cycle k of which all event times are known now becomes $k'-1$.
Now consider the lower graph of Figure 3-3. Suppose that a disturbance occurs in cycle k' at time t^*. Since the actual operation of the network will deviate from the desired operation, the optimization is started. However, the optimization must be restricted, as event times for which holds that $t_2 \leq \tilde{y}(k,t) \leq t^*$ are not allowed to be changed. Denote by *fixed* event times the event times which are not allowed to be varied in the optimization. All the other event times are called *variable* event times. The value $\tilde{y}_i(k \mid t)$ is an estimate resulting from an

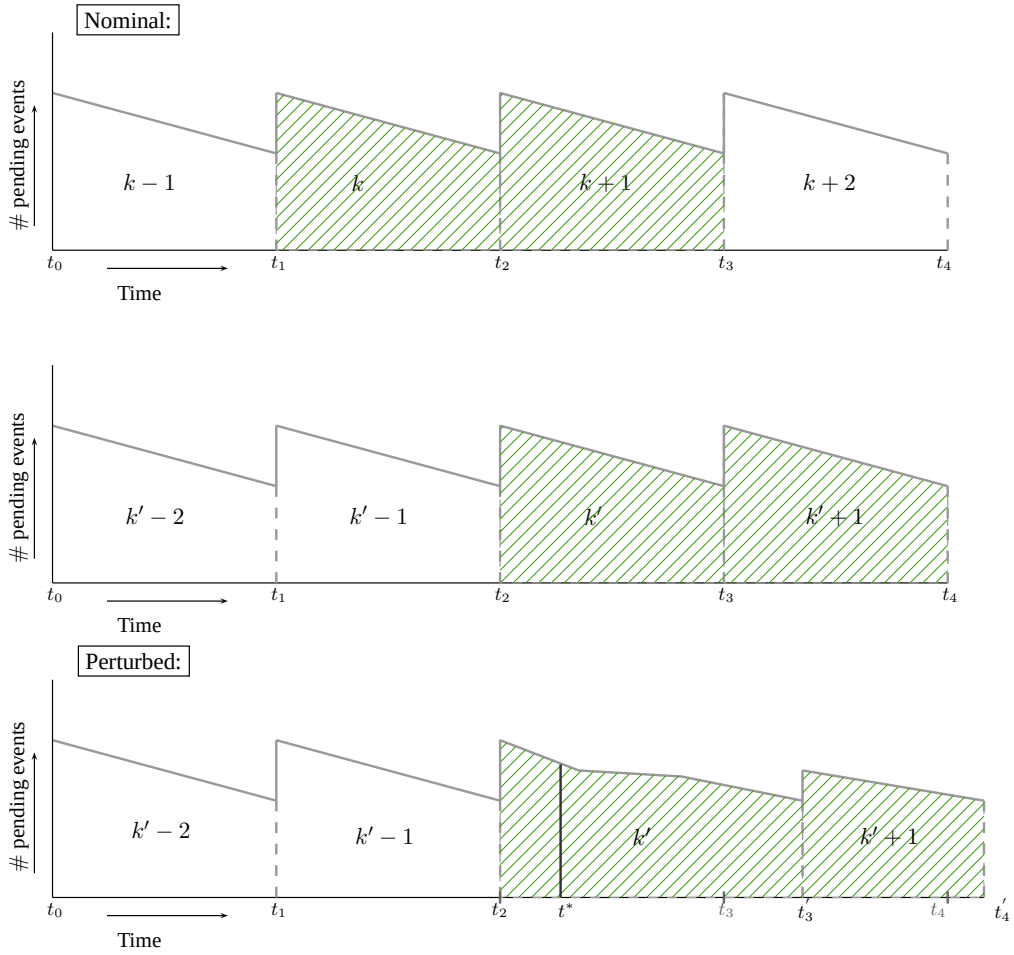

Figure 3-3: During disturbances events are delayed and it takes longer before all event information of the perturbed cycle is available, so at some point in time in that cycle, there are more pending events than there would be in the nominal case. Due to the disturbance, not all events of cycle k' have occurred until time $t_3^{'} > t_3$. As the disturbance also affects events from cycle $k'+1$, the point t_4 is also shifted to $t_4^{'}$. Note, however, that due to a stable timetable and effect of control actions, the distance between t_4 and $t_4^{'}$ is smaller than the distance between t_3 and $t_3^{'}$ (assuming no more disturbances occur).

estimated activity time $\tilde{\Theta}_{\gamma(i)}(k \mid t)$. Define the fixed and variable part of $\tilde{y}_i(k \mid t)$ as follows:

$$\tilde{y}_i^{\text{fix}}(k,t) = \tilde{y}_i(k,t) \quad \forall \quad \{i \mid t \geq \tilde{y}_i(k \mid t)\} \tag{3-55}$$

$$\tilde{y}_i^{\text{var}}(k,t) = \tilde{y}_i(k,t) \quad \forall \quad \{i \mid t < \tilde{y}_i(k \mid t)\} \tag{3-56}$$

When at time t an estimate of a perturbed parameter is supplied, the optimization is started. Like event times that occurred before time t, train orders cannot be changed when one of the trains has already departed or arrived. Therefore, all inputs associated to fixed events are not allowed to be changed in the optimization. However, more restrictions apply to order swaps, due to the time needed to perform the optimization. This is explained next.

3-5-2 Constraints due to computation time

Denote the time needed for the optimization to complete by δ_t. This implies that the optimal order swaps and event times are available no earlier than at time $t + \delta_t$. However, the optimization has the freedom to perform order swaps such that events can be re-scheduled to occur before $t + \delta_t$. When this is the case, at time $t + \delta_t$ it turns out that a train should have already departed for the order swap to be optimal. We should therefore give re-scheduled events a lower bound on event time, viz. $t + \delta_t$. This restriction is called the *minimum re-schedule time constraint*. First, we need to determine which inputs are not allowed to be changed by the optimization. Consider Figure 3-4 in which two events, event 1 and event 2, are to expected to occur at times $\tilde{y}_1$ and $\tilde{y}_2$, respectively. However, at time t_1 a new estimate of $\tilde{y}_1$ is available, predicting event 1 will occur at time $\tilde{y}_1(t_1)$. This is depicted in Figure 3-4. Event 2 was scheduled to occur after event 1, but due to perturbed event 1, it is better to let event 2 precede event 1. After the optimization at time $t + \delta_t$, two situations as depicted in Figure 3-4 could occur. Ideally, the departure time of train 2 equals its scheduled time again. However, the decision should be based on the departure time of train 2 not earlier than at $t+\delta_t$, since that is the time at which the order swap will be known. The optimization introduces some delay due to the relatively late decision. However, the optimal solution is not known beforehand. In the worst case, the situation would be the same as when no decision would have been made (lower graph of Figure 3-4.

The next example shows when it is not desired to change a train order. Consider the situation as depicted in Figure 3-5. A new parameter estimation is available at time t_1, which results in a new estimate $\tilde{y}_1(k \mid t_1)$. Assume that the order of train 1 and 2 can be changed. By the time the optimization routine has finished, the optimal decision might have been to change the order of train 1 and 2. In that case, train 2 should have left at time $\tilde{y}_2(t_1^-)$. However, this does not comply with the actual network situation, in which train 1 has already departed and train 2 is still waiting until enough headway time is ensured. Therefore, during the optimization the order between event 1 and 2 must be maintained. Moreover, if the order were allowed to be changed, both trains would need to wait for the decision to be computed. Both trains would then also be subjected to the minimum re-schedule time constraint of $t+\delta_t$. This would only increase the delay of both trains, which is not desired.

From the previous examples it can be concluded that we need to fix some of the inputs associated to future events. For that we need knowledge about the network situation at $t+\delta_t$, but this network situation changes due to the order swaps resulting from the optimization. Therefore, at the moment disturbance information comes available, the network is simulated with the perturbed parameter vector, while using the last known optimal input vector. All events that were fixed, remain fixed. Then, all inputs are fixed which are associated to the simulated event times smaller than $t + \delta_t$. The fixed and variable events are defined as:

$$\tilde{u}^{\text{fix}}(k,t) = \tilde{u}_{\alpha_{(i,\cdot)}}(k,t) \quad \forall \quad \{i \mid \tilde{y}_i(k \mid t) < t + \delta_t\} \tag{3-57}$$

$$\tilde{u}^{\text{var}}(k,t) = \tilde{u}_{\alpha_{(i,j)}}(k,t) \quad \forall \quad \{i \mid \tilde{y}_i(k \mid t) \geq t + \delta_t, \tilde{y}_j(k \mid t) \geq t + \delta_t\}, \tag{3-58}$$

where $\tilde{y}(k \mid t)$ is the prediction of the network situation at time t, based on the last parameter estimation and the last optimal input vector.

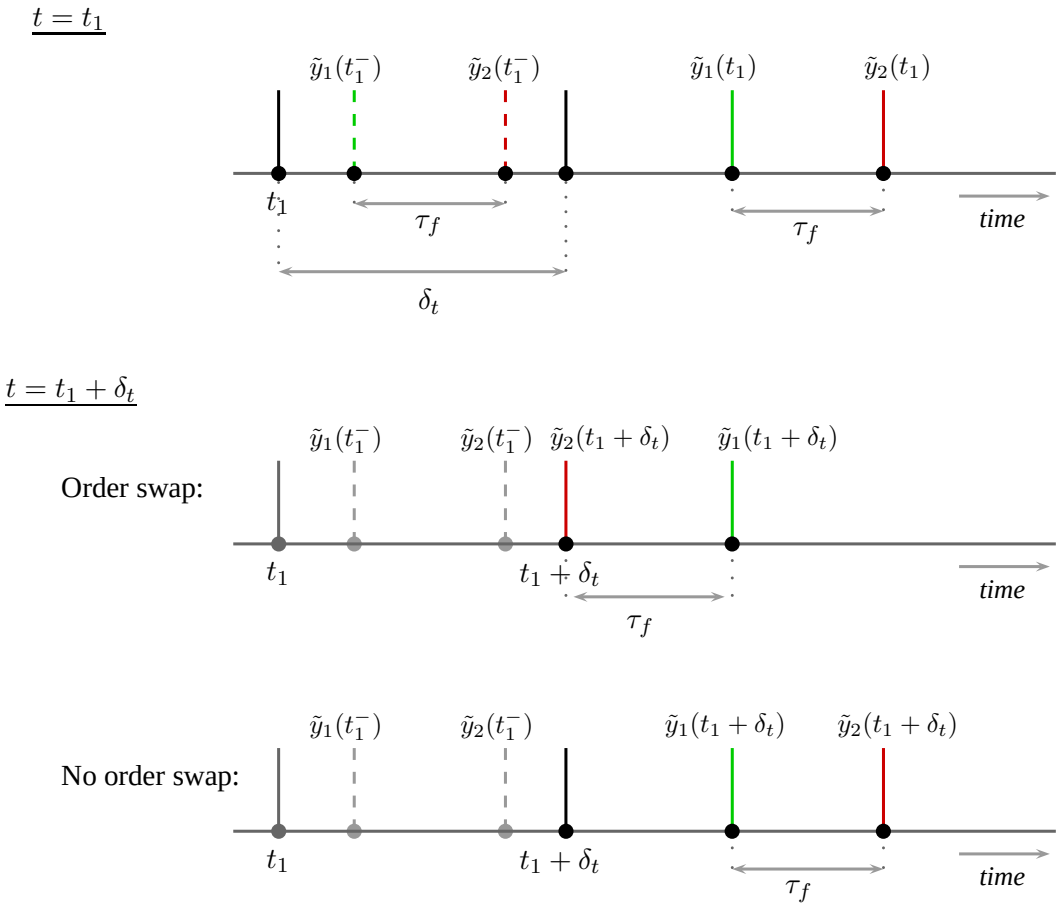

Figure 3-4: The decision to change the order of train 1 and 2 is not known until time $t + \delta_t$. Therefore, train 2 cannot have departed or arrived earlier than that time.

All events, which are allowed to be re-scheduled must have a minimum event time of $t + \delta_t$. The minimum re-schedule time constraint is given by Eq. (3-59).

$$\tilde{y}_i(k \mid t + \delta_t) \geq t + \delta_t \quad \forall \quad \{i \mid \tilde{y}_i(k \mid t) \geq t + \delta_t\} \tag{3-59}$$

Note that events for which holds that $\tilde{y}_i(k \mid t) \geq t + \delta_t$, may still be re-scheduled to a point in time between t and $t + \delta_t$. This can only happen to events which have a scheduled time $\tilde{s}_i$ between t and $t + \delta_t$, so it is sufficient to apply the constraint only to those events. Therefore, Eq. (3-59) is rewritten into:

$$\tilde{y}_i(k \mid t + \delta_t) \geq t + \delta_t \quad \forall \quad \{i \mid \tilde{y}_i(k \mid t) \geq t + \delta_t, \tilde{s}_i(k) < t + \delta_t\} \tag{3-60}$$

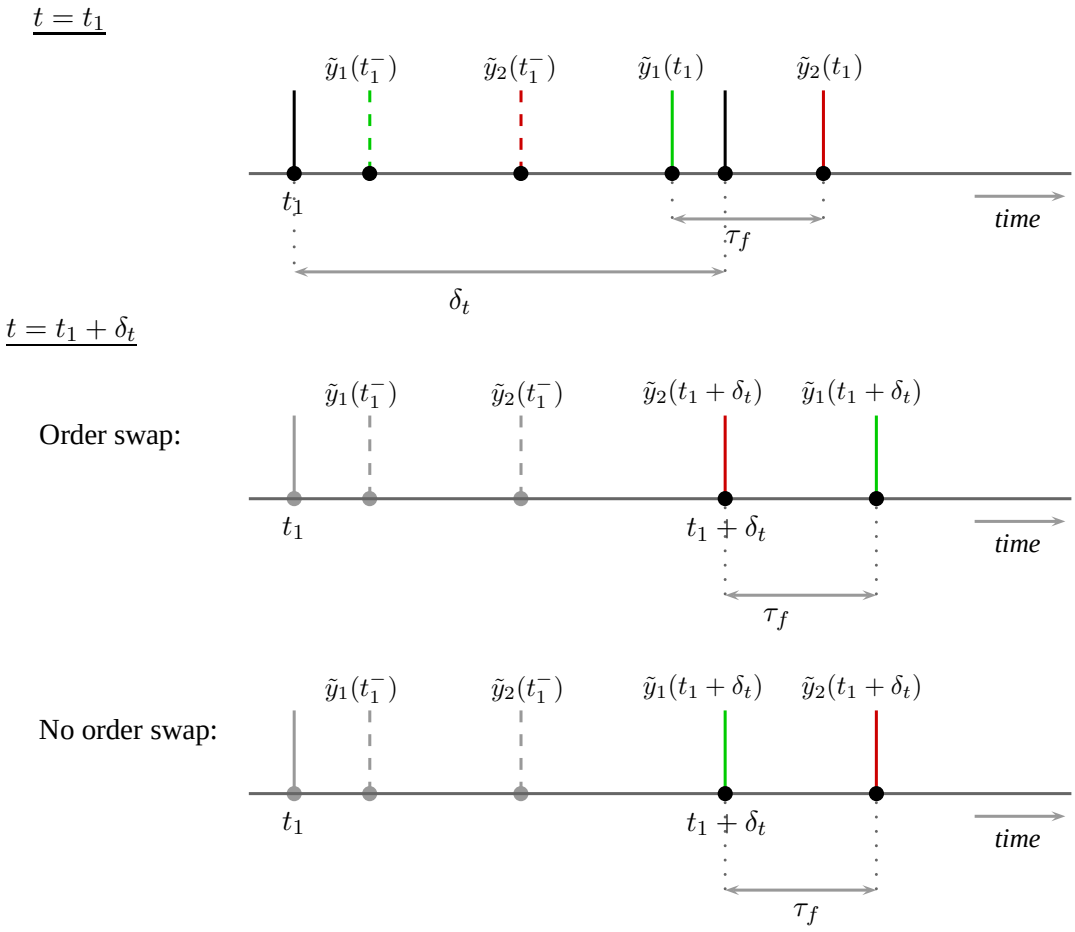

Figure 3-5: Situation in which the order must be maintained during the optimization. If an order swap were allowed, both events would have to be postponed to not earlier than $t + \delta_t$. In that case, the total delay would be bigger than when the order swap was not allowed.

Control horizon

The set of variable inputs can be reduced by introducing a control horizon. The control horizon can be defined over a fixed number of events or a fixed time interval. In the first case, the time this horizon spans is variable, while in the latter the number of events is variable. Defining the control horizon over a fixed time interval is preferred, since limiting the number of events may exclude some are estimated to occur later than δ_t and thus may still be manipulated now, but must be fixed the next optimization step. In this way, the controller never had the chance to manipulate inputs related to those events and may yield suboptimal solutions. One could state that simply making the control horizon large enough will circumvent this problem, but using a fixed time span provides a far more intuitive and better operation. The control horizon is thus defined over a fixed time interval N_c and is implemented including only inputs associated to variable events *scheduled* before $t + \delta_t + N_c$. A visualization of the control horizon is given by Figure 4-7, where t^* denotes $t + \delta_t$.

The control horizon is implemented by fixing control variables at their nominal values (i.e. 0)

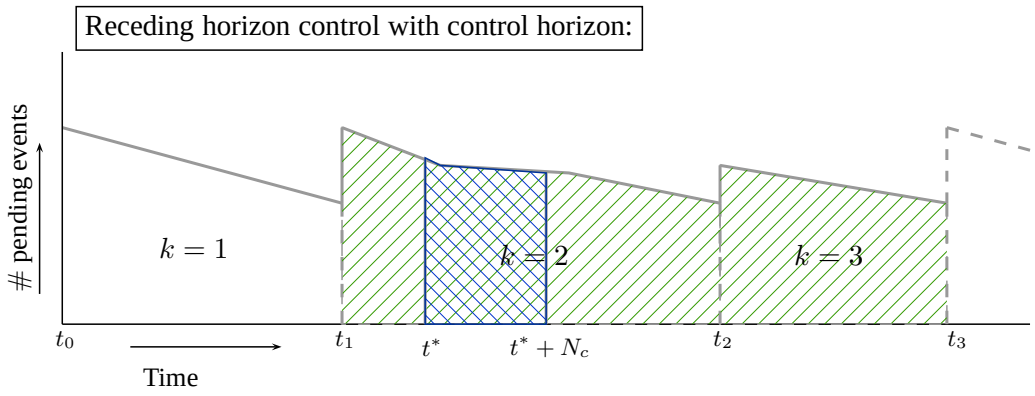

Figure 3-6: Visualization of the control horizon.

if they are associated to events beyond $t+\delta_t+N_c$. The fixed and variable part of $\tilde{u}(k,t)$ are then given by:

$$\tilde{u}^{\text{fix}}(k,t) = \tilde{u}_{\alpha_{(i,\cdot)}}(k,t) \quad \forall \quad \{i \mid \tilde{s}_i(k \mid t) < t+\delta_t \ \vee \ \tilde{s}_i(k \mid t) > t+\delta_t+N_c\}, \tag{3-61}$$

$$\tilde{u}^{\text{var}}(k,t) = \tilde{u}_{\alpha_{(i,j)}}(k,t) \tag{3-62}$$

$$\forall \quad \{i \mid \tilde{y}_i(k \mid t) \geq t+\delta_t, \tilde{s}_i(k \mid t) \leq t+\delta_t+N_c, \tilde{y}_j(k \mid t) \geq t+\delta_t, \tilde{s}_j(k \mid t) \leq t+\delta_t+N_c\}$$

3-5-3 Receding horizon MILP constraints

The optimizers of the MILP thus only need to consist of the variable event times and inputs. The constraints in Eq. (3-54) will be rewritten such that only $\tilde{y}^{\text{var}}(k \mid t)$ and $\tilde{u}^{\text{var}}(k,t)$ will be subjected to constraints. Partition $\tilde{y}(k \mid t)$ and $\tilde{u}(k,t)$ as follows:

$$\tilde{y}(k \mid t) = \begin{bmatrix} \tilde{y}^{\text{var}}(k \mid t) \\ \tilde{y}^{\text{fix}}(k,t) \end{bmatrix}, \quad \text{and} \quad \tilde{u}(k,t) = \begin{bmatrix} \tilde{u}^{\text{var}}(k,t) \\ \tilde{u}^{\text{fix}}(k,t) \end{bmatrix} \tag{3-63}$$

Using Eq. (3-63) we can rewrite Eq. (3-51) into:

$$J(k) = \begin{bmatrix} c_y^{\text{var}} \\ c_y^{\text{fix}} \end{bmatrix}^T \begin{bmatrix} \tilde{y}^{\text{var}}(k \mid t) \\ \tilde{y}^{\text{fix}}(k,t) \end{bmatrix} + \begin{bmatrix} c_u^{\text{var}} \\ c_u^{\text{fix}} \end{bmatrix}^T \begin{bmatrix} \tilde{u}^{\text{var}}(k,t) \\ \tilde{u}^{\text{fix}}(k,t) \end{bmatrix}. \tag{3-64}$$

Since $\tilde{y}^{\text{fix}}(k)$ and $\tilde{u}^{\text{fix}}(k)$ are constants, they can be removed from the cost function, yielding:

$$J(k) = [c_y^{\text{var}}]^T \tilde{y}^{\text{var}}(k \mid t) + [c_u^{\text{var}}]^T \tilde{u}^{\text{var}}(k,t). \tag{3-65}$$

The objective thus becomes:

$$\min_{\tilde{y}^{\text{var}}(k|t), \tilde{u}^{\text{var}}(k,t)} J(k) \tag{3-66}$$

Inserting Eq. (3-63) into the constraints as given by Eq. (3-54) gives:

$$\left[\begin{array}{cc} A_{c,\tilde{y}}^{\text{var}} & A_{c,\tilde{y}}^{\text{fix}} \end{array}\right] \left[\begin{array}{c} \tilde{y}^{\text{var}}(k \mid t) \\ \tilde{y}^{\text{fix}}(k,t) \end{array}\right] + \left[\begin{array}{cc} A_{c,\tilde{u}}^{\text{var}} & A_{c,\tilde{u}}^{\text{fix}} \end{array}\right] \left[\begin{array}{c} \tilde{u}^{\text{var}}(k,t) \\ \tilde{u}^{\text{fix}}(k,t) \end{array}\right] \geq b_c(k). \tag{3-67}$$

Bringing the constraints related to $\tilde{y}^{\text{fix}}(k)$ and $\tilde{u}^{\text{fix}}(k)$ to the right-hand side of Eq. (3-67), we obtain the following constraints:

$$\left[\begin{array}{cc} A_{c,\tilde{y}}^{\text{var}} & A_{c,\tilde{u}}^{\text{var}} \end{array}\right] \left[\begin{array}{c} \tilde{y}^{\text{var}}(k \mid t) \\ \tilde{u}^{\text{var}}(k,t) \end{array}\right] \geq b_c(k) - \left[\begin{array}{cc} A_{c,\tilde{y}}^{\text{fix}} & A_{c,\tilde{u}}^{\text{fix}} \end{array}\right] \left[\begin{array}{c} \tilde{y}^{\text{fix}}(k \mid t) \\ \tilde{u}^{\text{fix}}(k,t) \end{array}\right] \tag{3-68}$$

Eq. (3-68) needs to be augmented with the minimum re-schedule constraint:

$$\left[\begin{array}{cc} A_{c,\tilde{y}}^{\text{var}} & A_{c,\tilde{u}}^{\text{var}} \\ A_{rc} & 0 \end{array}\right] \left[\begin{array}{c} \tilde{y}^{\text{var}}(k \mid t) \\ \tilde{u}^{\text{var}}(k,t) \end{array}\right] \geq \left[\begin{array}{c} b_c(k) \\ t + \delta_t \end{array}\right] - \left[\begin{array}{cc} A_{c,\tilde{y}}^{\text{fix}} & A_{c,\tilde{u}}^{\text{fix}} \\ 0 & 0 \end{array}\right] \left[\begin{array}{c} \tilde{y}^{\text{fix}}(k \mid t) \\ \tilde{u}^{\text{fix}}(k,t) \end{array}\right] \tag{3-69}$$

3-6 Conclusions

This chapter presented a railway network prediction model, which has been structured with respect to the input and parameter vector. The headway time matrix of this prediction model can be manipulated by the control variable $\tilde{u}(k,t)$. The control of a railway network by changing train orders, can be written in a form which is affine in $\tilde{u}(k,t)$. The problem of finding the optimal set of event times and inputs, is written as an MILP, with continuous event times and binary inputs. To keep the number of constraints low, the optimization is carried out solely over the unknown event times and inputs. The ingredients for receding horizon control of railway networks have been derived, which are summarized below.

- All event times of the initial cycle are known. The horizon recedes one cycle ahead when all event times of the current cycle are known.
- Event times in the past will remain fixed.
- Inputs associated to event times between t and $t + \delta_t$ will be fixed.
- Event times are subjected to minimum re-schedule time due to this computation time.
- A control horizon is defined over inputs associated to variable event times scheduled before $t + \delta_t + N_c$. All other inputs are fixed at 0.
- The optimizer consist of only the variable event times and inputs.

Chapter 4

Implementation

4-1 Introduction

This chapter presents the implementation of the receding horizon control algorithm. The implementation is split into two parts, viz. building the system description given in Section 3-2 and the implementation of the receding horizon controller as presented in Section 3-5. The process of building the system model can be done offline, while the receding horizon control algorithm is an online process. The system description itself does not change during control, only the system states, inputs and parameters do. Parameter estimates are supplied to the controller, which computes optimal event times and inputs based on this parameter information. This is depicted in Figure 4-1.

The implementation is done in Mathworks Matlab, however the algorithms presented in this chapter are written in pseudo-code. These pseudo-codes use the format as shown in Algorithm 1.

This chapter is outlined as follows. First, the data structures used to represent the timetable and system model are presented in Section 4-2. Then, the algorithms for building the system matrices are treated in Section 4-3, after which Section 4-4 presents the various steps of the receding horizon control algorithm.

4-2 Data Structures

This section presents the data formats as used for the implementation of the receding horizon algorithm. For example, the system matrices $\mathcal{G}(\tilde{u}(k,t), \tilde{\Theta}(k \mid t))$ and $\mathcal{H}(u(k-1), \tilde{\Theta}(k \mid t))$ are generally very large for railway systems, but they are also sparse. Therefore, in stead of storing these matrices including mostly entries equal to ε, a more efficient representation is used. In the following, the arguments of the system matrices are left out, so for example $\mathcal{G}(\tilde{u}(k,t), \tilde{\Theta}(k \mid t))$ is written as $\mathcal{G}$. As the matrices $\mathcal{G}$ and $\mathcal{H}$ result from a timetable, the timetable format is treated first. How exactly these matrices are obtained from the timetable is covered in Section 4-3.

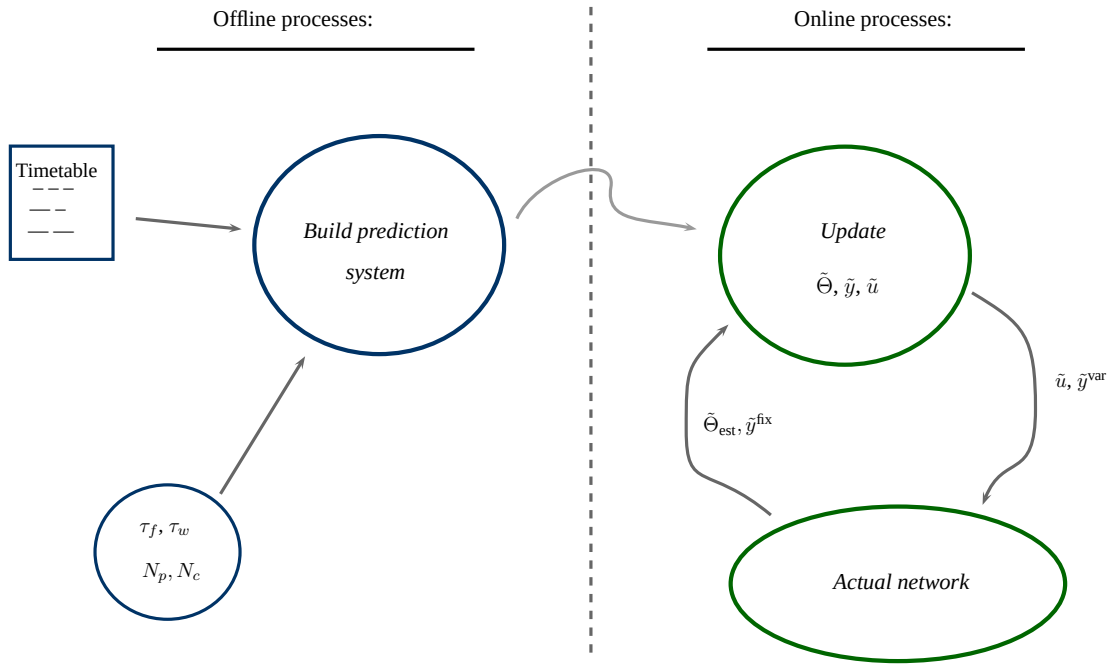

Figure 4-1: Global scheme of the algorithm.

Algorithm 1 Pseudo-code format.

Syntax:

How to use the function

Output:

List of output variables

Input:

List of input variables

Here come the steps that constitute the algorithm:

1. Step 1 *//Comments are written in italic light-grey text, following a '//'-sign*

   ⋮

n. Step n

4-2-1 Timetable

The nominal operation of the network is given by the timetable. For every train in the timetable, running and dwell times are specified. Furthermore, each train is associated to a *train line* and a *track number*, uses a track in a specific *direction* from which it should depart at its scheduled *departure time* and uses the track for a certain time given by the *running time*. The relation to preceding trains is given by the *previous train number* for the current cycle or previous cycle indicated by *cycle*. Finally, each train has a *dwell time*. The timetable

$tt = [train$	$train\ line$	$track$	$dir.$	$dep.\ time$	$run.\ time$	$prev.\ train$	$cycle$	$dwell\ time$	$arr.\ time]$

Table 4-1: The timetable format.

is stored as a matrix tt with the following structure:

Each row in this matrix is corresponds to a train. To meaning of each column is explained below.

Train The first column contains the train number. Recall that each train number represents a run on a track and thus multiple train numbers are associated to one physical train.

Train line Each train is associated to a specific train line, of which the service is carried out by one physical train.

Track This field assigns the train to a track in the network. This could either be a real track or a virtual track.

Dir. The direction of the train. Single tracks carry trains in opposite directions. This field has the value 0 or -1. The value should be chosen such that trains using same track in the same direction receive the same direction value.

Dep. time The scheduled departure time of the train.

Run. time The *minimal* running time of the train.

Prev. train The train number this physical train had on the previous track.

Cycle This field has the value 0 when the previous train is from the same cycle as this train, or -1 when from the previous cycle.

Dwell time The *minimal* time in minutes the previous train waited at a station or junction.

Arr. time The scheduled arrival time at the end-point of the track.

4-3 From Timetable to Prediction System

The system matrices describe the system operation within the prediction horizon and follow from the timetable. However, these matrices are sparse, i.e. most entries are equal to ε, and will take up a lot of computer memory to store. Therefore, to ensure efficient memory usage and reduce computational times, these matrices are converted to coordinate lists, which store

matrices as [*row*, *column*, *value*]. For matrices $\mathcal{G}$ and $\mathcal{H}$ extra columns are used to allow for the implementation of the inputs. The coordinate list has the following entries:

$$[row \mid column \mid nominal\ value \mid additional\ value \mid input\ \#]$$

When an input is active, *additional value* is added to *nominal value*. In the following, indices are no longer given in the subscript, but are shown between parentheses after the variable in question. For example, $\tilde{y}_i$ is written as $\tilde{y}(i)$.

4-3-1 Obtaining the running and dwell time matrices

Parameter vector $\tilde{\Theta}$ is stored separately, so the value entry in the coordinate list is replaced by the index of $\tilde{\Theta}$ at which the parameter can be found. As no inputs are associated to running and dwell times, matrices $\mathcal{A}_3$, $\mathcal{B}_3$ and $\mathcal{A}_4$ are stored as:

$$[row \mid column \mid \Theta_{\text{ind}} \mid 0 \mid 0]$$

The running time matrix $\mathcal{G}_4$ is implemented through constructing G_4. The steps needed to create G_4 are given in Algorithm 2. The first column of G_4 contains the index of the *arrival* time in $\tilde{y}(k,t)$, the second column contains the index of the *departure* time in $\tilde{y}(k,t)$ and the third column contains the index of $\tilde{\Theta}(k \mid t)$ (representing $\gamma(i)$ from Section 2-3-2). Recall that the fourth and fifth column are equal to 0, as no permutation is associated to a running time.

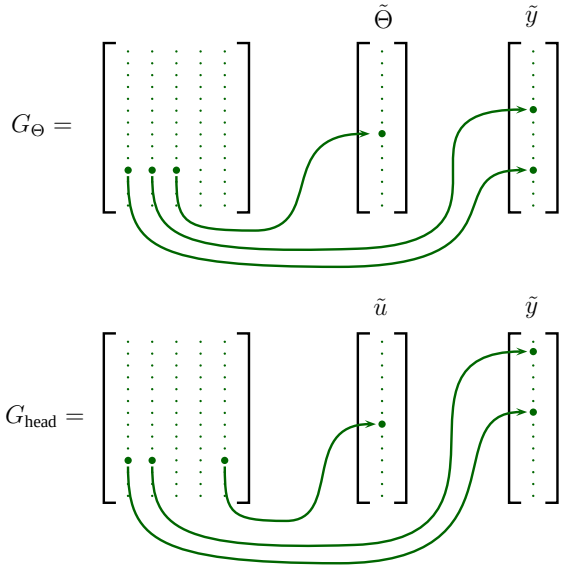

Figure 4-2: Entries of the coordinate lists point to entries in $\tilde{y}$, $\tilde{u}$ and $\tilde{\Theta}$.

The dwell time matrices are implemented through construction of G_3 and H_3. Denote the latest cycle of which all events are known by $k-1$. This is called the *initial cycle* and event times of this cycle are denoted by y_{init}. The *current cycle* is defined as the cycle following directly after the initial cycle. All dwell activities between trains from cycle k, the current

Algorithm 2 Obtaining $\mathcal{G}_4$.

Syntax:

$G_4 = \text{buildRun}(tt, n, N_p)$

Output:

G_4 *//Implementation of the running time matrix* $\mathcal{G}_4$

Input:

tt *//The timetable.*

n *//The number of trains per cycle.*

N_p *//The prediction horizon.*

1. $G_4 = \emptyset$
2. **For** $0 \leq k \leq N_p$
3. **For** $1 \leq i \leq n$ *//For each train in cycle 'k'*
4. $\tilde{\Theta}_{\text{ind}} = \tilde{\Theta}_{\text{ind}} + 1$ *//Increase index in* $\tilde{\Theta}$
5. $\tilde{\Theta}(\tilde{\Theta}_{\text{ind}}) = \text{tt(i ,6)}$ *//Add running time to* $\tilde{\Theta}$
6. $G_4 = \begin{bmatrix} G_4 \\ j + k \cdot n + n \cdot N_p & i + k \cdot n & \tilde{\Theta}_{\text{ind}} & 0 & 0 \end{bmatrix}$ *//Augment* G_4
7. **End**
7. **End**

cycle, and trains from cycle $k-1$, the initial cycle, are contained in H_3. All dwell activities between trains from cycle $k+\ell$ and $k+\ell+1$, where $0 \leq \ell \leq N_p - 1$ are contained in G_3. Algorithm 3 shows the procedure of creating G_3 and H_3. Figure 4-2 provides an illustration of the implementation of the running and dwell time matrices.

The running and dwell time matrices G_4, G_3 and H_3 will from now on be represented by one matrix, viz. G_Θ, containing G_4 and G_3, and H_Θ, containing H_3.

4-3-2 Obtaining the headway time matrices

Matrices $\mathcal{G}_1$,$\mathcal{H}_1$, $\mathcal{G}_2$ and $\mathcal{H}_2$ are stored as H_1, H_2, G_1 and G_2, respectively, in the following format:

$$[i \mid j \mid \tau^{\text{nom}} \mid \tau_{\text{alt}} - \tau_{\text{nom}} \mid \tilde{u}_{\text{ind}}],$$

where τ_{nom} is the nominal headway time between event j and i, and $\tau_{\text{alt}} - \tau_{\text{nom}}$ is added to τ_{nom} when input $\tilde{u}(u_{\text{ind}})$ is active. The input vector is stored separately and u_{ind} gives the index of the associated input in $\tilde{u}$. Note that τ_{nom} can be β, τ_f or τ_w. The same holds for τ^{alt}. Algorithms 4 and 5 show the steps that constitute to construction of H_1, H_2, G_1 and G_2.

4-4 Receding Horizon Algorithm

The receding horizon algorithm is executed along four main steps: Disturbance information can be supplied at various time instants (step 2). The states and inputs are separated into fixed and variable parts (steps 3 and 4) and the subsystem related to the variable states is extracted (step 5). The optimization (step 8) runs when new information becomes available, when the system has shifted ahead one cycle or, when using a control horizon, at every time step. The optimization has a dual function: to provide a simulation of the network and to find the optimal event times and inputs. As soon as all event times of the current cycle k are known, the system is shifted one cycle ahead and the variables are updated (step 11).

Step 3 of Algorithm 6 is illustrated in Figure 4-3. At each time step t a vector $\tilde{y}$ with estimated event times is obtained. Part of this vector contains event times prior to time t. These event times lie in the past and are not allowed to be changed in the optimization routine. The indices in $\tilde{y}$ of those event times are contained in the set $\tilde{y}^{\text{fix}}_{\text{ind}}$. All event times greater than the current time t are contained in the set $\tilde{y}^{\text{var}}_{\text{ind}}$.

Step 4 of Algorithm 6 is illustrated in Figure 4-4. Here, the set of inputs is divided into two subsets $\tilde{u}^{\text{var}}$ and $\tilde{u}^{\text{fix}}$.

Only the variable event times and inputs need to be optimized. Therefore, the optimization only requires the entries of G related to the variable event times. These entries constitute the matrix G^{var} and is formed by all rows of G of which the first column is in the set $\tilde{y}^{\text{var}}_{\text{ind}}$. Figure 4-5 visualizes this process.

The algorithms used for each step are treated separately next. In the following, the arguments k and t are dropped for presentational reasons only.

Algorithm 3 Obtaining $\mathcal{G}_3$ & $\mathcal{H}_3$.

Syntax:

$G_3, H_3 = \text{buildDwell}(tt, n, N_p)$

Output:

G_3 *//Implementation of the dwell time matrix* $\mathcal{G}_3$
H_3 *//Implementation of the dwell time matrix* $\mathcal{H}_3$

Input:

tt *//The timetable.*
n *//The number of trains per cycle.*
N_p *//The prediction horizon.*

1. $G_3 = \emptyset$, $H_3 = \emptyset$
2. **For** $0 \leq \ell \leq N_p$ *//For each cycle*
3. **For** $1 \leq i \leq n$ *//For each train in one cycle*
4. j = tt(i,7) *//The previous train*
5. $\tilde{\Theta}_{\text{ind}} = \tilde{\Theta}_{\text{ind}} + 1$ *//Increase index in* $\tilde{\Theta}$
6. $\tilde{\Theta}(\tilde{\Theta}_{\text{ind}})$ = tt(i ,9) *//Augment* $\tilde{\Theta}$ *with the dwell time*
7. **If** tt(i,8)== 0 *//The previous train is from the same cycle*
8. $G_3 = \begin{bmatrix} G_3 \\ i + \ell \cdot n & j + \ell \cdot n + n \cdot N_p & \tilde{\Theta}_{\text{ind}} & 0 & 0 \end{bmatrix}$ *//Augment* G_3
9. **Else** *//The previous train is from the previous cycle*
10. **If** $\ell == 0$ *//The previous train is from* $k-1$*, so constraint is contained in* $\mathcal{H}_3$
11. $H_3 = \begin{bmatrix} H_3 \\ i & j + n & \tilde{\Theta}_{\text{ind}} & 0 & 0 \end{bmatrix}$ *//Augment* H_3
12. **Else** *//Constraint is contained in* $\mathcal{G}_3$
13. $G_3 = \begin{bmatrix} G_3 \\ i + \ell \cdot n & j + (\ell - 1) \cdot n + n \cdot N_p & \tilde{\Theta}_{\text{ind}} & 0 & 0 \end{bmatrix}$ *//Augment* G_3
14. **End**
15. **End**
16. **End**
17. **End**

Algorithm 4 Obtaining $\mathcal{H}_1$ & $\mathcal{H}_2$.

Syntax:

$H_1, H_2 = \text{buildHhead}(tt, n, N_p, \tau_f, \tau_w)$

Output:

H_{head} *//Implementation of the headway time matrices $\mathcal{H}_1$ and $\mathcal{H}_2$*

Input:

tt *//The timetable.*

n *//The number of trains per cycle.*

N_p *//The prediction horizon.*

τ_f *//Headway time for trains running in the same direction.*

τ_w *//Headway time for trains running in the opposite direction.*

1. $H_1^p = \emptyset$, $H_2^p = \emptyset$, $H_{\text{head}} = \emptyset$
2. **For** $1 \leq p \leq n_t$ *//For each track*
3. Sort trains on track p with respect to departure time
4. **For** each train i in cycle k using track p
5. **For** each train j in cycle $k-1$ using track p
6. **If** direction of i is the same as that of j
7. **If** train j is not the previous train from $k-1$ for train i
8. $u_{\text{ind}} = u_{\text{ind}} + 1$ *//Add input/Increase index of input vector*
9. $H_1^p = \begin{bmatrix} & & H_1^p & & \\ i & j & \tau_f & \beta - \tau_f & u_{\text{ind}} \\ i+n & j+n & \tau_f & \beta - \tau_f & u_{\text{ind}} \end{bmatrix}$ *//Augment H_1^p*
10. **Else** *//trains travel in opposite directions*
11. $u_{\text{ind}} = u_{\text{ind}} + 1$ *//Add input/Increase index of input vector*
12. $H_2^p = \begin{bmatrix} & H_2^p & & \\ i & j+n & \tau_w & \beta - \tau_w & u_{\text{ind}} \end{bmatrix}$ *//Augment H_2^p*
13. **End** *//End direction condition*
14. **End** *//End "for each train j"*
15. **End** *//End "for each train i"*
16. $H_{\text{head}} = \begin{bmatrix} H_{\text{head}} \\ H_1^p \\ H_2^p \end{bmatrix}$ *//Augment H_{head} with track matrices H_1^p and H_2^p*
17. **End** *//End "for each track p"*

Algorithm 5 Obtaining $\mathcal{G}_1$ & $\mathcal{G}_2$.

Syntax:

$G_1, G_2 = \text{buildGhead}(tt, n, N_p, \tau_f, \tau_w)$

Output:

G_{head} *//Implementation of the headway time matrix* $\mathcal{G}_1$ *and* $\mathcal{G}_2$

Input:

tt *//The timetable.*

n *//The number of trains per cycle.*

N_p *//The prediction horizon.*

τ_f *//Headway time for trains running in the same direction.*

τ_w *//Headway time for trains running in the opposite direction.*

1. $G_1^p = \emptyset$, $G_2^p = \emptyset$, $G_{\text{head}} = \emptyset$
2. **For** $1 \leq p \leq n_t$ *//For each track*
3. Sort all trains using track p in prediction horizon with respect to departure time
4. **For** each train i using track p
5. **For** each train j using track p
6. $u_{\text{ind}} = u_{\text{ind}} + 1$ *//Add input/Increase index of input vector*
7. **If** direction of i is the same as that of j
8. $$G_1^p = \begin{bmatrix} & & G_1^p & & \\ i & j & \tau_f & \beta - \tau_f & u_{\text{ind}} \\ i + n \cdot N_p & j + n \cdot N_p & \tau_f & \beta - \tau_f & u_{\text{ind}} \\ j & i & \beta & \tau_f - \beta & u_{\text{ind}} \\ j + n \cdot N_p & i + n \cdot N_p & \beta & \tau_f - \beta & u_{\text{ind}} \end{bmatrix} \text{ //Augment } G_1^p$$
9. **Else** *//trains travel in opposite directions*
10. $$G_2^p = \begin{bmatrix} & G_2^p & & & \\ i & j + n \cdot N_p & \tau_w & \beta - \tau_w & u_{\text{ind}} \\ j & i + n \cdot N_p & \beta & \tau_w - \beta & u_{\text{ind}} \end{bmatrix} \text{ //Augment } G_2^p$$
11. **End** *//End direction condition*
12. **End** *//End "for each train j"*
13. **End** *//End "for each train i"*
14. $$G_{\text{head}} = \begin{bmatrix} G_{\text{head}} \\ G_1^p \\ G_2^p \end{bmatrix} \text{ //Augment } G_{head} \text{ with track matrices } G_1^p \text{ and } G_2^p$$
15. **End** *//End "for each track p"*

Algorithm 6 Receding horizon algorithm outline.

1. **For** each timestep t
2. Update $\tilde{y}(k \mid t), \tilde{\Theta}(k \mid t)$
3. separate variable and fixed event times
4. separate variable and fixed inputs
5. extract subsystem related to variable event times
6. **If** new parameter information or system has shifted or when using control horizon
7. Build constraints related to variable states
8. simulate/find optimal inputs and states
9. **End**
10. **If** current cycle k known
11. Update variables
12. **End**
13. **End**

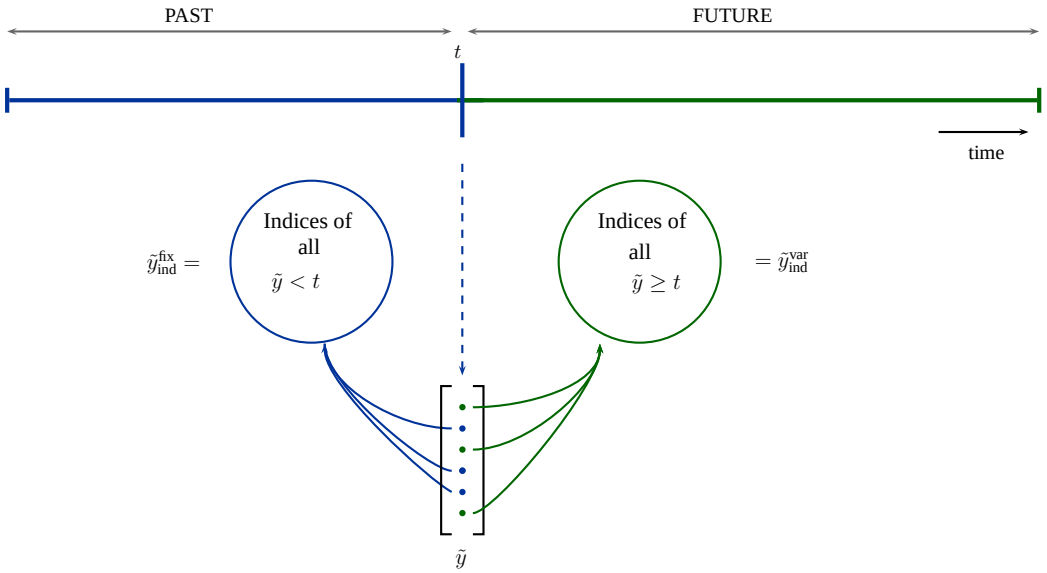

Figure 4-3: Separation of fixed and variable event times.

4-4-1 Supplying disturbance information

At various time steps, estimates on perturbed parameters may come available. These estimates are contained in a list Z, with the following format for each row i:

$$Z_i = [\tilde{\Theta}_{\text{ind}} \mid \tilde{\Theta}_{\text{est}}(\tilde{\Theta}_{\text{ind}}) \mid t_{\text{est}}],$$

where $\tilde{\Theta}_{\text{ind}}$ is the index in $\tilde{\Theta}$ of the perturbed parameter and $\tilde{\Theta}_{\text{est}}(\tilde{\Theta}_{\text{ind}})$ is the estimated parameter value at time t_{est}. During the simulation at time t_{est}, the associated parameter is updated with the estimate. After each parameter update, the system is simulated whit this

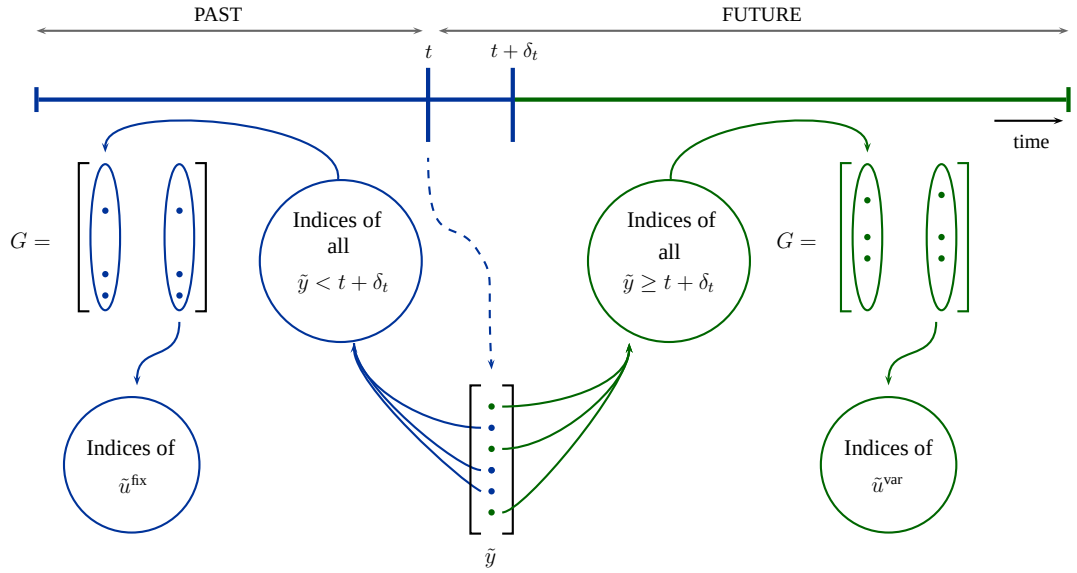

Figure 4-4: Separation of fixed and variable inputs.

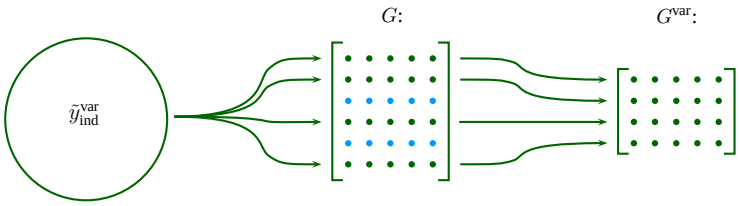

Figure 4-5: Extracting the sub system related to $\tilde{y}^{\text{var}}$.

new parameter vector, while keeping the inputs at their last optimal value. This simulation was carried out by running the optimization with all the inputs fixed.

4-4-2 Fixing the states and inputs

The states and inputs are fixed according to the rules (Eq. (3-57), Eq. (3-57), Eq. (3-55) and Eq. (3-56)) presented in Section 3-5. Steps 2 and 3 of Algorithms 8 and 9 rely on *logical indexing*. The logical test $\tilde{y} \leq t$ results in a binary vector with a length equal to that of $\tilde{y}$, with entries equal to 1 at the indices where $\tilde{y} \leq t$ is true. When the vector y^{mask} is used as index for 'yLinList', all the entries of 'yLinList' at the indices where $y^{\text{mask}} = 1$ are obtained.

4-4-3 Extracting sub-matrices

After finding the fixed and variable states, the entries from G and H related to the *variable* states can be extracted, such that only the constraints related to those states are used in the MILP. This procedure is shown in Algorithm 10. The fixed states are constants and do not depend on other events any longer. Therefore, constraints yielding the event time of an already fixed state can be discarded. Again, logical indexing is used in steps 7 to 17. The "**length**()" operator gives the length of a vector, so for a vector v of size 1×6, **length**(v)

Algorithm 7 Updating $\tilde{\Theta}$ with available disturbance information.

Syntax:

$\tilde{\Theta} = \text{getDistInfo}(Z, G_\Theta, t, \delta_t)$

Output:

$\tilde{\Theta}$ *//Updated parameter vector.*

Input:

$\tilde{\Theta}$*//Previous parameter vector.*

Z *//Matrix with disturbance information.*

1. **If** $Z\,! = \emptyset$ *//The '!' denotes the 'not'-operator.*
2. $Z_{\text{ind}} =$**find**$(Z(:,3) == t)$
3. **End**
4. **If** $Z_{\text{ind}}\,! = \emptyset$*//If Z contains information at time t.*
5. **For** j=1:length(Z_{ind})*//For each parameter perturbation at time t.*
6. $\tilde{\Theta}(Z(Z_{\text{ind}}(j),1)) := Z(Z_{\text{ind}}(j),2)$ *//Update $\tilde{\Theta}$*
7. **End**
8. **End**

yields 6. Since also only the parameters related to variable states are of interest, a reduced parameter vector $\tilde{\Theta}_{var}$ is created. The indices in column 3 of G_{var} and H_{var} (denoted by $G_{\text{var}}(:,3)$ and $H_{\text{var}}(:,3)$, respectively) are changed to refer to $\tilde{\Theta}_{var}$, instead of $\tilde{\Theta}$. This is done in steps 19 to 24.

4-4-4 Constructing the MILP constraints

The next step to be carried out is the optimization. First, the state and input indices in G_{var} and H_{var} need to be mapped to the correct indices in $\tilde{u}_{\text{var}}$, $\tilde{y}_{\text{var}}$, $\tilde{y}_{\text{fix}}$ and $\tilde{u}_{\text{fix}}$. Then the constraints have to be constructed. The MILP is solved using the MILP solver as part of the MPT [26] toolbox available for MATLAB.

Index mapping

Since the first two columns of G_{var} and H_{var} point to entries in the full state vector, these indices need to be mapped to point to the correct index in $\tilde{u}_{\text{var}}$. This is also the case for the fifth column (containing the input indices) of G_{var} and H_{var}. Therefore, the mapping vectors $\tilde{y}^{\text{fix}}_{\text{map}}$, $\tilde{y}^{\text{var}}_{\text{map}}$, $\tilde{u}^{\text{fix}}_{\text{map}}$ and $\tilde{u}^{\text{var}}_{\text{map}}$ are created, according to the steps as shown in Algorithms 11 and 12. Note that the indices of these mapping vectors are 1 higher than that of $\tilde{u}$ and $\tilde{y}$. This is due to the fact that MATLAB does not allow zero-indexing, so a 1 has to be reserved to indicated a non-used entry, instead of a 0. The concept of index mapping is visualized in Figure 4-6.

Algorithm 8 Separation of fixed and variable states.

Syntax:

$[y^{\text{mask}}, \tilde{y}^{\text{fix}}_{\text{ind}}, \tilde{y}^{\text{var}}_{\text{ind}}, \tilde{y}^{\text{fix}}, \tilde{y}^{\text{var}}] = \text{fixStates}(\tilde{y}, t, n, N_p)$

Output:

y^{mask} *//A binary mask with 1's on the places corresponding to fixed events.*

$\tilde{y}^{\text{fix}}_{\text{ind}}$ *//Indices of fixed events.*

$\tilde{y}^{\text{var}}_{\text{ind}}$ *//Indices of variable events.*

$\tilde{y}^{\text{fix}}$ *//The fixed event times.*

$\tilde{y}^{\text{var}}$ *//The variable event times.*

Input:

$\tilde{y}$ *//The states.*

t *//The current time.*

n, N_p *//The number of trains per cycle and the length of the prediction horizon.*

//Obtain a vector with entries linearly ascending in value from 1 to $2 \cdot n \cdot N_p$*:*

1. $\text{yLinList} = [1, 2, \ldots, 2 \cdot n \cdot N_p]^T$
2. $y^{\text{mask}} = (\tilde{y} < t)$ *//Binary vector* $y^{mask}(i) = 1$ *if* $\tilde{y}(i) < t$
3. $\tilde{y}^{\text{fix}}_{\text{ind}} = \text{yLinList}(y^{\text{mask}})$ *//Result: vector with indices of fixed* $\tilde{y}$*.*
4. $\tilde{y}^{\text{var}}_{\text{ind}} = \text{yLinList}(!y^{\text{mask}})$ *//*$\tilde{y}^{var}_{ind}$ *contains indices of variable* $\tilde{y}$ *('!' is the 'not'-operator.)*
5. $\tilde{y}^{\text{fix}} = \tilde{y}(\tilde{y}^{\text{fix}}_{\text{ind}})$ *//The values of the fixed states.*
6. $\tilde{y}^{\text{var}} = \tilde{y}(\tilde{y}^{\text{var}}_{\text{ind}})$ *//The values of the variable states.*

Algorithm 9 Separation of fixed and variable inputs:

Syntax:

$[\tilde{u}^{\text{fix}}_{\text{ind}}, \tilde{u}^{\text{var}}_{\text{ind}}, \tilde{u}^{\text{fix}}, \tilde{u}^{\text{var}}] = \text{fixInputs}(\tilde{u}, G_{\text{head}}, \tilde{y}, t, \delta_t, n_{\tilde{u}})$

Output:

$\tilde{u}^{\text{fix}}_{\text{ind}}$, $\tilde{u}^{\text{var}}_{\text{ind}}$ *//Indices of fixed and variable inputs.*
$\tilde{u}^{\text{fix}}$,$\tilde{u}^{\text{var}}$ *//Values of fixed and variable inputs.*

Input:

$\tilde{u}$ *//The input vector.*
G_{head} *//The headway time matrix* G_{head}
$\tilde{y}$ *//The state vector*
t *//The actual time.*
δ_t *//The time needed for optimization.*
$n_{\tilde{u}}$ *//The total number of inputs within the prediction horizon.*

1. $\text{uLinList} = [1, 2, \ldots, n_{\tilde{u}}]^T$ *//Vector with entries linearly ascending in value from 1 to* $n_{\tilde{u}}$*:*
2. $\text{yLinList} = [1, 2, \ldots, n \cdot N_p]^T$ *//Vector with entries linearly ascending in value from 1 to* $n \cdot N_p$*:*
3. $y^{\text{mask}} = (\tilde{y} < t + \delta_t)$ *//Binary vector* $y^{mask}(i) = 1$ *if* $\tilde{y}(i) < t + \delta_t$
4. $\tilde{y}^{\text{fix}}_{\text{ind}} = \text{yLinList}(y^{\text{mask}})$*//Result: vector with indices of fixed* $\tilde{y}$*.*
5. $\tilde{u}^{\text{fix}}_{\text{ind}} := \emptyset$
6. **For** $1 \leq j \leq$ **length**$(\tilde{y}^{\text{fix}}_{\text{ind}})$
7. $\quad \text{maskG} = \left(G_{\text{head}}(:, 1) == \tilde{y}^{\text{fix}}_{\text{ind}}(j)\right)$
8. $\quad \tilde{u}^{\text{fix}}_{\text{ind}} := \begin{bmatrix} \tilde{u}^{\text{fix}}_{\text{ind}} \\ G_{\text{head}}(\text{maskG}, 5) \end{bmatrix}$
9. $\quad G_{\text{head}} := G_{\text{head}}(!\text{maskG}, :)$ *//Remove already evaluated entries of* G_{head}*.*
10. **End**
11. $\tilde{u}^{\text{var}}_{\text{ind}} = \text{uLinList} \setminus \tilde{u}^{\text{fix}}_{\text{ind}}$*//All indices of variable* $\tilde{y}$
12. $\tilde{u}^{\text{var}} = \tilde{u}(\tilde{u}^{\text{var}}_{\text{ind}})$
13. $\tilde{u}^{\text{fix}} = \tilde{u}(\tilde{u}^{\text{fix}}_{\text{ind}})$

Algorithm 10 Extracting sub-matrices related to variable states.

Syntax:

$[G_{\text{var}}, H_{\text{var}}, \tilde{\Theta}_{\text{var}}] = \text{extractSubSys}(G_{\text{head}}, H_{\text{head}}, G_{\Theta}, H_{\Theta}, \tilde{y}^{\text{var}}_{\text{ind}}, \tilde{\Theta})$

Output:

$G_{\text{var}},\ H_{\text{var}}$ *//The submatrices related to* $\tilde{y}^{var}$

$\tilde{\Theta}_{\text{var}}$ *//The parameter vector.*

Input:

$G_{\text{head}},\ H_{\text{head}},\ G_{\Theta},\ H_{\Theta}$

$\tilde{y}^{\text{var}}_{\text{ind}}$ *//Indices of variable events.*

$\tilde{\Theta}$ *//The parameter vector.*

1. $G_{\text{var,head}} = \emptyset,\ G_{\text{var},\Theta} = \emptyset$
2. $G_{\text{temp,head}} = G_{\text{head}},\ H_{\text{temp,head}} = H_{\text{head}}$*//Store* G *and* H *in a temporary array for manipulation:*
3. $G_{\text{temp},\Theta} = G_{\Theta},\ H_{\text{temp},\Theta} = H_{\Theta}$
4. **For** $1 \leq i \leq$ **length**$(\tilde{y}^{\text{var}}_{\text{ind}})$

   //Find all entries of G *associated to* $\tilde{y}^{var}_{ind}(i)$*:*
5. $G_{\text{ind,head}} = \left(G_{\text{temp,head}}(:,1) == \tilde{y}^{\text{var}}_{\text{ind}}(i)\right)$ *//*$G_{ind,head}$ *is a binary mask.*
6. $G_{\text{ind},\Theta} = \left(G_{\text{temp},\Theta}(:,1) == \tilde{y}^{\text{var}}_{\text{ind}}(i)\right)$ *//*$G_{ind,\Theta}$ *is a binary mask.*
7. $G_{\text{var,head}} = \begin{bmatrix} G_{\text{var,head}} \\ G_{\text{temp,head}}(G_{\text{ind}},:) \end{bmatrix},\quad G_{\text{var},\Theta} = \begin{bmatrix} G_{\text{var},\Theta} \\ G_{\text{temp},\Theta}(G_{\text{ind}},:) \end{bmatrix}$ *//Augment* $G_{var,head}$ *and* $G_{var,\Theta}$*:*

   //We do not need to evaluate these entries anymore:
8. $G_{\text{temp,head}} = G_{\text{temp,head}}(!G_{\text{ind,head}},:),\ G_{\text{temp},\Theta} = G_{\text{temp},\Theta}(!G_{\text{ind},\Theta},:)$

   //Find all entries of H *associated to* $\tilde{y}^{var}_{ind}(i)$*:*
9. $H_{\text{ind,head}} = \left(H_{\text{temp,head}}(:,1) == \tilde{y}^{\text{var}}_{\text{ind}}(i)\right)$ *//*$H_{ind,head}$ *is a binary mask.*
10. $H_{\text{ind},\Theta} = \left(H_{\text{temp},\Theta}(:,1) == \tilde{y}^{\text{var}}_{\text{ind}}(i)\right)$ *//*$H_{ind,\Theta}$ *is a binary mask.*
11. $H_{\text{var,head}} = \begin{bmatrix} H_{\text{var,head}} \\ H_{\text{temp,head}}(H_{\text{ind}},:) \end{bmatrix},\quad H_{\text{var},\Theta} = \begin{bmatrix} H_{\text{var},\Theta} \\ H_{\text{temp},\Theta}(H_{\text{ind}},:) \end{bmatrix}$ *//Augment* $H_{var,head}$ *and* $H_{var,\Theta}$*:*

    //We do not need to evaluate these entries anymore:
12. $H_{\text{temp,head}} = H_{\text{temp,head}}(!H_{\text{ind,head}},:),\ H_{\text{temp},\Theta} = H_{\text{temp},\Theta}(!H_{\text{ind},\Theta},:)$
13. **End**

    //Create $\tilde{\Theta}_{var}$ *and let* $G_{var,\Theta}(:,3)$ *and* $H_{var,\Theta}(:,3)$ *point to indices in* $\tilde{\Theta}_{var}$*:*
14. **For** each row j in $G_{\text{var},\Theta}$
15. $\tilde{\Theta}_{var}(j) = \tilde{\Theta}(G_{\text{var},\Theta}(j,3))$*//Construct parameter vector associated to variable states.*
26. $G_{\text{var},\Theta}(j,3) = j$*//Let* $G_{var,\Theta}(j,3)$ *point to the index in* $\tilde{\Theta}_{var}$*.*
17. **End**
18. $c =$ **length**$(\tilde{\Theta}_{var})$
19. **For** each row j in $H_{\text{var},\Theta}$
20. $\tilde{\Theta}_{var}(j+c) = \tilde{\Theta}(H_{\text{var},\Theta}(j,3))$*//Construct parameter vector associated to variable states.*
21. $H_{\text{var},\Theta}(j,3) = j + c$*//Let* $H_{var,\Theta}(j,3)$ *point to the index in* $\tilde{\Theta}_{var}$*.*
22. **End**.
23. $G_{\text{var}} = \begin{bmatrix} G_{\text{var},\Theta} \\ G_{\text{var,head}} \end{bmatrix},\ H_{\text{var}} = \begin{bmatrix} H_{\text{var},\Theta} \\ H_{\text{var,head}} \end{bmatrix}$

Algorithm 11 State mapping.

Syntax:

$[\tilde{y}^{\text{var}}_{\text{map}}, \tilde{y}^{\text{fix}}_{\text{map}}] = \text{mapStates}(\tilde{y}^{\text{fix}}, \tilde{y}^{\text{var}}, \tilde{y}^{\text{fix}}_{\text{ind}}, \tilde{y}^{\text{var}}_{\text{ind}}, n, N_p)$

Output:

$\tilde{y}^{\text{var}}_{\text{map}}, \tilde{y}^{\text{fix}}_{\text{map}}$

Input:

$\tilde{y}^{\text{fix}}, \tilde{y}^{\text{var}}$

$\tilde{y}^{\text{fix}}_{\text{ind}}, \tilde{y}^{\text{var}}_{\text{ind}}$

n, N_p *//The number of trains per cycle and the length of the prediction horizon.*

1. $\tilde{y}^{\text{fix}}_{\text{map}}(i) = \textbf{ones}(2 \cdot n \cdot N_p, 1)$*//Initial value: a* $1 \times 2 \cdot n \cdot N_p$ *vector of 1's.*
2. $\tilde{y}^{\text{var}}_{\text{map}}(i) = \textbf{ones}(2 \cdot n \cdot N_p, 1)$*//Initial value: a* $1 \times 2 \cdot n \cdot N_p$ *vector of 1's.*
3. $\text{num1} = [1, \ldots, \textbf{length}(\tilde{y}^{\text{fix}})]$ *//num1 contains integer values from 1 to* ***length***$(\tilde{y}^{fix})$
4. $\text{num2} = [1, \ldots, \textbf{length}(\tilde{y}^{\text{var}})]$ *//num2 contains integer values from 1 to* ***length***$(\tilde{y}^{var})$
5. $\tilde{y}^{\text{fix}}_{\text{map}}(\tilde{y}^{\text{fix}}_{\text{ind}} + 1) = \text{num1} + 1$
6. $\tilde{y}^{\text{var}}_{\text{map}}(\tilde{y}^{\text{var}}_{\text{ind}} + 1) = \text{num2} + 1$

Algorithm 12 Input mapping.

Syntax:

$[\tilde{u}^{\text{var}}_{\text{map}}, \tilde{u}^{\text{fix}}_{\text{map}}] = \text{mapInputs}(\tilde{u}^{\text{fix}}, \tilde{u}^{\text{var}}, \tilde{u}^{\text{fix}}_{\text{ind}}, \tilde{u}^{\text{var}}_{\text{ind}}, n, N_p)$

Output:

$\tilde{u}^{\text{var}}_{\text{map}}, \tilde{u}^{\text{fix}}_{\text{map}}$

Input:

$\tilde{u}^{\text{fix}}, \tilde{u}^{\text{var}}$

$\tilde{u}^{\text{fix}}_{\text{ind}}, \tilde{u}^{\text{var}}_{\text{ind}}$

n, N_p *//The number of trains per cycle and the length of the prediction horizon.*

1. $\tilde{u}^{\text{fix}}_{\text{map}}(i) = \textbf{ones}(2 \cdot n \cdot N_p, 1)$ *//Initial value: a* $1 \times 2 \cdot n \cdot N_p$ *vector of 1's.*
2. $\tilde{u}^{\text{var}}_{\text{map}}(i) = \textbf{ones}(2 \cdot n \cdot N_p, 1)$*//Initial value: a* $1 \times 2 \cdot n \cdot N_p$ *vector of 1's.*
3. $\text{num1} = [1, \ldots, \textbf{length}(\tilde{u}^{\text{fix}})]$ *//num1 contains integer values from 1 to* ***length***$(\tilde{u}^{fix})$
4. $\text{num2} = [1, \ldots, \textbf{length}(\tilde{u}^{\text{var}})]$ *//num2 contains integer values from 1 to* ***length***$(\tilde{u}^{var})$
5. $\tilde{u}^{\text{fix}}_{\text{map}}(\tilde{u}^{\text{fix}}_{\text{ind}} + 1) = \text{num1} + 1$
6. $\tilde{u}^{\text{var}}_{\text{map}}(\tilde{u}^{\text{var}}_{\text{ind}} + 1) = \text{num2} + 1$

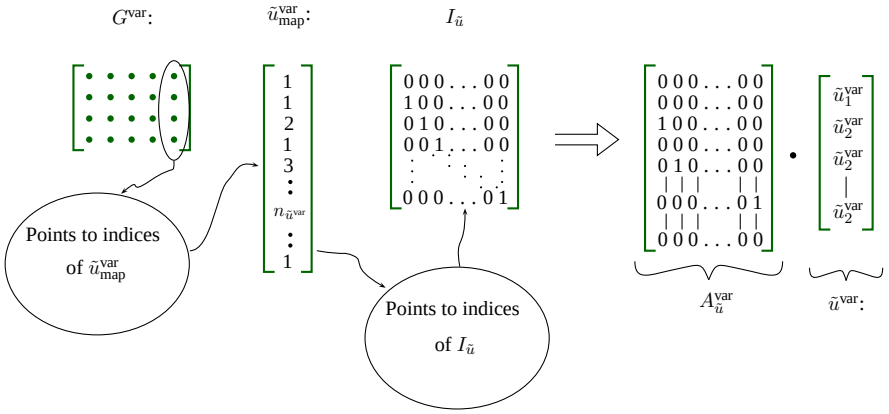

Figure 4-6: Visualization of index mapping. The same applies to the event time vector

Constraints

The final step in the optimization procedure, is to obtain the constraints as given by Eq. (3-68). The constraints and the optimizers are supplied separately to the MILP solver and we thus end up with a matrix A_c and a vector b_c. Matrix A_c is multiplied with the optimizer. The constraints follow directly from G_{var} and H_{var}. However, they must be translated from a coordinate list into a matrix, suitable for multiplication. First of all, the first column of G_{var} contains an index of $\tilde{y}$. We need the index of $\tilde{y}^{\text{var}}$, so by using $\tilde{y}^{\text{var}}_{\text{map}}(G_{\text{var}}(:,1)+1)$ we obtain the indices in $\tilde{y}^{\text{var}}$. This index is then used to select a row of an identity matrix I_G, such that the correct entry is selected when multiplying this row with $\tilde{y}^{\text{var}}$. The same is done for H_{var}, by using $\tilde{y}^{\text{var}}_{\text{map}}(H_{\text{var}}(:,1)+1)$ to select a row of an identity matrix I_H.
Selecting the correct entry of the input vector is done in a similar fashion, viz. by using $\tilde{u}^{\text{var}}_{\text{map}}$ and $I_{\tilde{u}}$. However, $G_{\text{var},\Theta}$ and $G_{\text{var},\Theta}$ refer to input index 0, which is non-existing in MATLAB. Therefore, to select the correct entry in $\tilde{u}_{\text{var}}$, 1 is added to the index and the identity matrix $I_{\tilde{u}}$ is augmented with a row of zeros at the top. In this way, when referring to index 0 in $\tilde{u}_{\text{var}}$, the *first* row $(0+1)$ of $I_{\tilde{u}}$ is selected, resulting in a multiplication of zeros with the input vector. Furthermore, I_{init} is an identity matrix of size $\mathbf{length}(y_{\text{init}})\times\mathbf{length}(y_{\text{init}})$, $I_{\tilde{s}}$ is an identity matrix of size $\mathbf{length}(\tilde{s})\times\mathbf{length}(\tilde{s})$, $I_{\tilde{y}}$ is an identity matrix of size $\mathbf{length}(\tilde{y}^{\text{var}})\times\mathbf{length}(\tilde{y}^{\text{var}})$, $I_{\tilde{y}^{\text{fix}}}$ is an identity matrix of size $\mathbf{length}(\tilde{y}^{\text{fix}})\times\mathbf{length}(\tilde{y}^{\text{fix}})$ and $I_{\tilde{u}^{\text{fix}}}$ is an identity matrix of size $\mathbf{length}(\tilde{u}^{\text{fix}})\times\mathbf{length}(\tilde{u}^{\text{fix}})$.
The solver used for solving the MILP requires the inequality constraints to be supplied in the form of $A\cdot z\leq b$, instead of $A\cdot z\geq b$ as was presented in Section 3-4. Therefore, both sides of the constraints given by (3-68) are multiplied by -1, as $-A\cdot z\leq -b$ is equivalent to $A\cdot z\geq b$.

4-4-5 Shifting the horizon

The moment that all events of the current cycle of the prediction system are known, the system switches to the next cycle. Recall that each cycle consists of n events. Suppose that the current cycle is denoted by k_1 and the cycle to shift to is denoted by k_2 $(=k_1+1)$. All events from cycle k_1 are known, so y_{init} will be equal to $y(k_1)$ (for now, k_1 is not an index here, but denotes the cycle). All other departure and arrival times shift n places up in the state vector and they will thus receive a new index in $\tilde{y}$ as well as a new input index. All

Algorithm 13 Construction of the MILP constraints.

Syntax:

$[A_c, b_c] = \text{buildConstraints}(G_{\text{var}}, H_{\text{var}}, \tilde{\Theta}_{\text{var}}, y_{\text{init}}, \tilde{y}^{\text{fix}}, \tilde{u}^{\text{fix}}, \tilde{y}^{\text{var}}_{\text{ind}}, \tilde{y}^{\text{var}}_{\text{map}}, \tilde{y}^{\text{fix}}_{\text{map}}, \tilde{u}^{\text{var}}_{\text{map}}, \tilde{u}^{\text{fix}}_{\text{map}}, t, \delta_t, \tilde{s})$

Output:

A_c *//The left-hand side of the constraints:* $A_c \cdot z \leq b_c$.

b_c *//The right-hand side of the constraints:* $A_c \cdot z \leq b_c$.

Input:

G_{var}, H_{var} *//The sub-matrices associated to the variable states.*

$\tilde{\Theta}_{\text{var}}$ *//The parameters associated to the variable states.*

y_{init} *//The initial states.*

$\tilde{y}^{\text{fix}}$, $\tilde{u}^{\text{fix}}$ *//The values of the fixed states and inputs.*

$\tilde{y}^{\text{var}}_{\text{ind}}$ *//The indices of the variable states.*

$\tilde{y}^{\text{var}}_{\text{map}}$, $\tilde{y}^{\text{fix}}_{\text{map}}$ *//State-mapping.*

$\tilde{u}^{\text{var}}_{\text{map}}$, $\tilde{u}^{\text{fix}}_{\text{map}}$ *//Input-mapping.*

$t, \delta_t, \tilde{s}$ *//The current time, the optimization time and the schedule.*

//Construction of A_c*:*

1. $A_{c1,\tilde{y}^{\text{var}}} = -I_G(\tilde{y}^{\text{var}}_{\text{map}}(G_{\text{var}}(:,1)+1),:) + I_G(\tilde{y}^{\text{var}}_{\text{map}}(G_{\text{var}}(:,2)+1),:)$
2. $A_{c1,\tilde{u}^{\text{var}}} = G_{\text{var}}(:,4) \cdot I_{\tilde{u}}(\tilde{u}^{\text{var}}_{\text{map}}(G_{\text{var}}(:,5)+1),:)$
3. $A_{c2,\tilde{y}^{\text{var}}} = -I_H(\tilde{y}^{\text{var}}_{\text{map}}(H_{\text{var}}(:,1)),:)$
4. $A_{c3,\tilde{y}^{\text{var}}} = -I_{\tilde{s}}$
5. $A_{rc} = I_{\tilde{y}}(y^{\tilde{\text{var}}} < t + \delta_t)$
6. $A_c = \begin{bmatrix} A_{c1,\tilde{y}^{\text{var}}} & A_{c1,\tilde{u}^{\text{var}}} \\ A_{c2,\tilde{y}^{\text{var}}} & \mathbf{0} \\ A_{c3,\tilde{y}^{\text{var}}} & \mathbf{0} \\ A_{rc} & \mathbf{0} \end{bmatrix}$

//Construction of b_c*:*

7. $b^{\text{var}}_{c1,\tilde{\Theta}} = -\tilde{\Theta}(G_{\text{var},\tilde{\Theta}}(:,3))$
8. $b^{\text{var}}_{c2,\tilde{\Theta}} = -I_{\text{init}}(H_{\text{var},\tilde{\Theta}}(:,2),:) \cdot y_{\text{init}} - \tilde{\Theta}(H_{\text{var},\tilde{\Theta}}(:,3))$
9. $b^{\text{var}}_{c1,\text{head}} = -G_{\text{var,head}}(:,3)$
10. $b^{\text{var}}_{c2,\text{head}} = -I_{\text{init}}(H_{\text{var,head}}(:,2),:) \cdot y_{\text{init}} - H_{\text{var,head}}(:,3)$
11. $b^{\text{var}}_{c3} = -\tilde{s}(\tilde{y}^{\text{var}}_{\text{ind}})$
12. $A_{c1,\tilde{y}^{\text{fix}}} = [I_{\tilde{y}^{\text{fix}}}(\tilde{y}^{\text{fix}}_{\text{map}}(G_{\text{var}}(:,2)+1))] \cdot \tilde{y}^{\text{fix}}$
13. $A_{c1,\tilde{u}^{\text{fix}}} = [G_{\text{var}}(:,4) \cdot I_{\tilde{u}^{\text{fix}}}(\tilde{u}^{\text{fix}}_{\text{map}}(G_{\text{var}}(:,5)+1))] \cdot \tilde{u}^{\text{fix}}$
14. $A_{c2,\tilde{u}^{\text{fix}}} = H_{\text{var}}(j,4) \cdot I_{\tilde{u}^{\text{fix}}}(\tilde{u}^{\text{fix}}_{\text{map}}(H_{\text{var}}(:,5)+1)) \cdot \tilde{u}^{\text{fix}}$
15. $b^{\text{var}}_{c1} = \begin{bmatrix} b^{\text{var}}_{c1,\tilde{\Theta}} \\ b^{\text{var}}_{c1,\text{head}} \end{bmatrix}, b_{c2} = \begin{bmatrix} b^{\text{var}}_{c2,\tilde{\Theta}} \\ b^{\text{var}}_{c2,\text{head}} \end{bmatrix}$
16. $b_c = \begin{bmatrix} b^{\text{var}}_{c1} - A_{c1,\tilde{y}^{\text{fix}}} - A_{c1,\tilde{u}^{\text{fix}}} \\ b^{\text{var}}_{c2} - A_{c2,\tilde{u}^{\text{fix}}} \\ b^{\text{var}}_{c3} \\ -(t + \delta_t) \end{bmatrix}$

events and inputs of cycle k_1 that were fixed, need to be remained fixed when progressing to cycle k_2. To ensure that the conditions imposed on the inputs in the new cycle are consistent with those from the previous cycle, a matrix is created which maps input indices from k_1 to the input indices as used in k_2. This is done in Algorithm 15. Inputs which were first assigned to $y(k_1)$ will now have to be mapped to input indices as found in H, since $y(k_1)$ has now become y_{init}. These mapping matrices can be created beforehand and can be used every time the system switches to a new cycle. The parameter vector is updated using Algorithm 7. Recall that the dwell times relating events from the initial cycle to events from the current cycle can still change and should therefore not be discarded.

4-4-6 Control Horizon

The control horizon is implemented by Algorithm 17. All inputs related to events having their scheduled time after $t + \delta_t + N_c$ are fixed to the default value of 0.

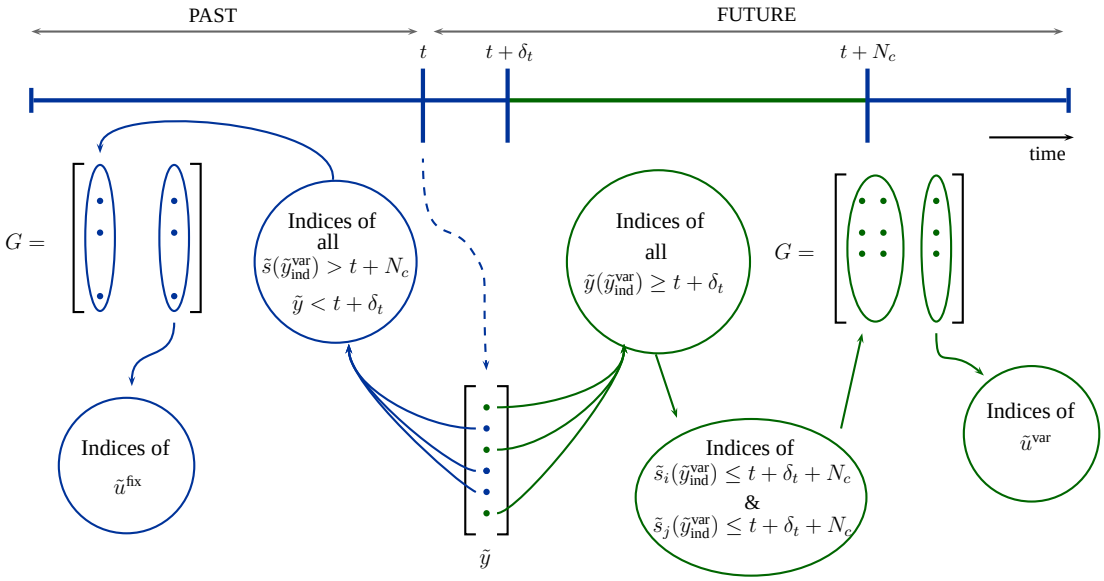

Figure 4-7: Separation of fixed and variable inputs when using a time-based control horizon.

4-5 Conclusions

An implementation of the receding horizon controller as presented in Section 3-5 was presented as well as the implementation of model as presented in Section 3-2. Every time step, the receding horizon controller splits the state vector and input vector into two subsets, viz. a set of fixed states corresponding to past events, a set of variable states corresponding to future states, a set of fixed inputs corresponding to past events, a set of variable inputs corresponding to future states. The large and sparse system matrices are implemented using a coordinate list, saving computer memory and computation time. This implementation is tested on a small network, for which some test cases have been designed. This is treated in the next chapter.

Algorithm 14 Variable updates.

Output:

y_{init}
y^{mask}
$\tilde{u}$
$y^{\text{fix}}_{\text{ind}}$, $u^{\text{fix}}_{\text{ind}}$
y^{fix}, u^{fix}

Input:

n, N_p *//The number of trains per cycle and the length of the prediction horizon.*
$\tilde{y}$ *//The state vector.*
y^{mask}, u^{mask} *//Binary fixed state and input mask*

//When all states of cycle k_1 are known:

1. **If** $y_i^{\text{mask}} == 1 \quad \forall \quad (1 \leq i \leq n)$ & $(N_p \cdot n + 1 \leq i \leq N_p \cdot n + n)$
2. $y_{\text{init}} = \begin{bmatrix} \tilde{y}(1:n) \\ \tilde{y}(N_p \cdot n + 1 : N_p \cdot n + n) \end{bmatrix}$ *//Update y_{init}.*
3. $\tilde{y}^{\text{fix}} = \tilde{y}(\begin{bmatrix} y^{\text{mask}}(n : N_p \cdot n) \\ y^{\text{mask}}(N_p \cdot n + 1 : 2 \cdot N_p \cdot n) \end{bmatrix})$ *//Update $\tilde{y}^{fix}$.*
4. $y^{\text{mask}} = \begin{bmatrix} y^{\text{mask}}(1 : N_p \cdot n - n) \\ 0 \\ y^{\text{mask}}(N_p \cdot n + n : 2 \cdot N_p \cdot n) \\ 0 \end{bmatrix}$ *//Update y^{mask} (0 is a vector of length n).*
5. $\text{linList} = [1, 2, \ldots, 2 \cdot n \cdot N_p]^T$
6. $y^{\text{fix}}_{\text{ind}} = \text{linList}(y^{\text{mask}})$
7. $\tilde{s} := \tilde{s} + T$ *//Update schedule.*
8. $\tilde{y} := \begin{bmatrix} \tilde{y}(n : n \cdot N_p - n) \\ \tilde{s}(n \cdot N_p - n + 1 : n \cdot N_p) \\ \tilde{y}(n \cdot N_p + 1 : 2 \cdot n \cdot N_p - n) \\ \tilde{s}(2 \cdot n \cdot N_p - n + 1 : 2 \cdot n \cdot N_p) \end{bmatrix}$ *//Update event times.*
9. $\tilde{u}, \tilde{u}^{\text{fix}}_{\text{ind}} = \text{shiftU}(\tilde{u}^{\text{fix}}_{\text{ind}}, \tilde{u}, G_{\text{head}}, H_{\text{head}}, n, N_p)$ *//Update $\tilde{u}$ and $\tilde{u}^{fix}_{ind}$.*
10. $\tilde{\Theta} = \text{shiftTheta}(\tilde{\Theta}, G_\Theta, H_\Theta, n, N_p)$ *//Update $\tilde{\Theta}$*

Algorithm 15 Input vector update on cycle shift.

Syntax:

$\tilde{u}^{\text{new}}, \tilde{u}^{\text{fix}}_{\text{ind}} = \text{shiftU}(\tilde{u}^{\text{fix}}_{\text{ind}}, \tilde{u}^{\text{prev}}, G_{\text{head}}, H_{\text{head}}, n, N_p)$

Output:

$\tilde{u}^{\text{new}}$

$\tilde{u}^{\text{fix}}_{\text{ind}}$

Input:

$\tilde{u}^{\text{fix}}_{\text{ind}}$ *//The previous indices of inputs to fix.*

$\tilde{u}^{\text{prev}}$ *//The previous input values.*

G_{head}, H_{head} *//The headway time matrices.*

n, N_p *//The number of trains per cycle and the length of the prediction horizon.*

//Find indices in G related to event times $[y(k+1)\ \ldots\ y(k+N_p)]$

//For presentational reasons, G_{head} *is written as G.*

1. $\text{index1} = (G(:,1) > n\ \&\ G(:,1) \leq N_p \cdot n\ \&\ G(:,2) > n\ \&\ G(:,2) \leq N_p \cdot n) \parallel$
   $(G(:,1) > n\ \&\ G(:,1) \leq N_p \cdot n\ \&\ G(:,2) > N_p \cdot n + n)$

//Find indices in G_{head} *related to event times* $[y(k)\ \ldots\ y(k+N_p-1)]$

2. $\text{index2} = (G(:,1) > 0\ \&\ G(:,1) \leq N_p \cdot n - n\ \&\ G(:,2) > 0\ \&\ G(:,2) \leq N_p \cdot n - n) \parallel$
   $(G(:,1) > 0\ \&\ G(:,1) \leq N_p \cdot n - n\ \&\ G(:,2) > N_p \cdot n\ \&\ G(:,2) \leq (2 \cdot N_p \cdot n - n))$

//Find indices in G relating event times $y(k+1)$ *and* $y(k)$*:*

3. $\text{index3} = (G(:,1) > n\ \&\ G(:,1) \leq 2 \cdot n\ \&\ (G(:,2) \leq n) \parallel$
   $(G(:,1) > n\ \&\ G(:,1) \leq 2 \cdot n\ \&\ (G(:,2) > N_p \cdot n\ \&\ G(:,2) \leq N_p \cdot n + n)$
4. $\tilde{u}^{\text{prev}}_{\text{ind}} = G(\text{index1}, 5)$ *//The input indices related to* $[y(k+1)\ \ldots\ y(k+N_p)]$*.*
5. $\tilde{u}^{\text{prev}}_{\text{ind}} := \textbf{unique}(\tilde{u}^{\text{prev}}_{\text{ind}})$ *//Remove repetions.*
6. $\tilde{u}^{\text{new}}_{\text{ind}} = G(\text{index2}, 5)$*//The input indices related to* $[y(k)\ \ldots\ y(k+N_p-1)]$*.*
7. $\tilde{u}^{\text{new}}_{\text{ind}} := \textbf{unique}(\tilde{u}^{\text{new}}_{\text{ind}})$*//Remove repetions.*
8. $\tilde{u}^{\text{H,fix}}_{\text{ind}} = \textbf{unique}(H(:,5))$ *//Find all input indices relating* $y(k)$ *and* y_{init}*.*
9. $\tilde{u}^{\text{mask}} = \textbf{zeros}(n_{\tilde{u}}, 1)$ *//Initialize binary mask*
10. $\tilde{u}^{\text{mask}}(\tilde{u}^{\text{fix}}_{\text{ind}}) = 1$ *//Binary mask for all fixed inputs in the previous cycle.*
11. $\tilde{u}^{\text{G,fix}}_{\text{ind}} = \tilde{u}^{\text{new}}_{\text{ind}}(\tilde{u}^{\text{mask}})$*//Indices the fixed inputs receive in the new cycle*
12. $\tilde{u}^{\text{fix}}_{\text{ind}} := \begin{bmatrix} \tilde{u}^{\text{H,fix}}_{\text{ind}} \\ \tilde{u}^{\text{G,fix}}_{\text{ind}} \end{bmatrix}$ *//All input indices to fix in the new cycle.*
13. $\tilde{u}^{\text{new}}(\tilde{u}^{\text{new}}_{\text{ind}}) = \tilde{u}^{\text{prev}}(\tilde{u}^{\text{prev}}_{\text{ind}})$ *//Update the input vector.*

Algorithm 16 Parameter vector update on cycle shift.

Syntax:

$\tilde{\Theta}^{\text{new}} = \text{shiftTheta}(\tilde{\Theta}^{\text{old}}, G_\Theta, H_\Theta, n, N_p)$

Output:

$\tilde{\Theta}^{\text{new}}$ *//The updated parameter vector with shifted entries.*

Input:

$\tilde{\Theta}^{\text{old}}$ *//The parameter vector for the previous cycle of the prediction system.*

$\tilde{\Theta}^{\text{nom}}$ *//The nominal parameter values.*

G_Θ, H_Θ *//The running and dwell time matrices*

n, N_p *//The number of trains per cycle and the length of the prediction horizon.*

//Find indices in G related to event times $[y(k+1)\ \ldots\ y(k+N_p)]$

1. $\text{index1} = (G(:,1) > n\ \&\ G(:,1) \leq N_p \cdot n\ \&\ G(:,2) > n\ \&\ G(:,2) \leq N_p \cdot n) \parallel$
$(G(:,1) > n\ \&\ G(:,1) \leq N_p \cdot n\ \&\ G(:,2) > N_p \cdot n + n)$

//Find indices in G related to event times $[y(k)\ \ldots\ y(k+N_p-1)]$

2. $\text{index2} = (G(:,1) > 0\ \&\ G(:,1) \leq N_p \cdot n - n\ \&\ G(:,2) > 0\ \&\ G(:,2) \leq N_p \cdot n - n) \parallel$
$(G(:,1) > 0\ \&\ G(:,1) \leq N_p \cdot n - n\ \&\ G(:,2) > N_p \cdot n\ \&\ G(:,2) \leq (2 \cdot N_p \cdot n - n))$

//Find indices in G relating event times $y(k+1)$ *and* $y(k)$*:*

3. $\text{index3} = (G(:,1) > n\ \&\ G(:,1) \leq 2 \cdot n\ \&\ (G(:,2) \leq n) \parallel$
$(G(:,1) > n\ \&\ G(:,1) \leq 2 \cdot n\ \&\ (G(:,2) > N_p \cdot n\ \&\ G(:,2) \leq N_p \cdot n + n)$

4. $\tilde{\Theta}^{\text{new}} = \tilde{\Theta}^{\text{nom}}$ *//Refresh parameter vector.*

//Shift the parameter values one cycle:

5. $\tilde{\Theta}^{\text{new}}(G_\Theta(\text{index2}, 3)) = \tilde{\Theta}^{\text{old}}(G_\Theta(\text{index1}, 3))$

6. $\tilde{\Theta}^{\text{new}}(H_\Theta(:, 3)) = \tilde{\Theta}^{\text{old}}(G_\Theta(\text{index3}, 3))$

Algorithm 17 Separation of fixed and variable inputs within control horizon:

Syntax:

$[\tilde{u}^{\text{fix}}_{\text{ind}}, \tilde{u}^{\text{var}}_{\text{ind}}, \tilde{u}^{\text{fix}}, \tilde{u}^{\text{var}}] = \text{fixInputs}(\tilde{u}, G_{\text{head}}, \tilde{y}, t, \delta_t, N_c, n_{\tilde{u}})$

Output:

$\tilde{u}^{\text{fix}}_{\text{ind}}$, $\tilde{u}^{\text{var}}_{\text{ind}}$ *//Indices of fixed and variable inputs.*
$\tilde{u}^{\text{fix}}$,$\tilde{u}^{\text{var}}$ *//Values of fixed and variable inputs.*

Input:

G_{head} *//The headway time matrix* G_{head}
$\tilde{y}$ *//The state vector*
t *//The actual time.*
δ_t *//The time needed for optimization.*
N_c *//The control horizon in minutes.*
N_p *//The prediction horizon.*
$n_{\tilde{u}}$ *//The total number of inputs within the prediction horizon.*

1. $\text{uLinList} = [1, 2, \ldots, n_{\tilde{u}}]^T$ *//Vector with integer values from 1 to* $n_{\tilde{u}}$*:*
2. $\text{yLinList} = [1, 2, \ldots, n \cdot N_p]^T$ *//Vector with integer values from 1 to* $n \cdot N_p$*:*
3. $\tilde{y}^{\text{var}}_{\text{ind}} = \text{yLinList} \setminus \tilde{y}^{\text{fix}}_{\text{ind}}$
4. $\tilde{y}^{\text{var}}_{\text{ind}} := \tilde{y}^{\text{var}}_{\text{ind}}(\tilde{s}(\tilde{y}^{\text{var}}_{\text{ind}}) \leq t + \delta_t + N_c)$
5. $G_{\text{temp1}} = \emptyset$
6. **For** $1 \leq j \leq$ **length**$(\tilde{y}^{\text{var}}_{\text{ind}})$
7. $\quad \text{maskG} = \left(G_{\text{head}}(:, 1) == \tilde{y}^{\text{var}}_{\text{ind}}(j)\right)$
8. $\quad G_{\text{temp1}} := \begin{bmatrix} G_{\text{temp1}} \\ G_{\text{head}}(\text{maskG}, :) \end{bmatrix}$
9. **End**
10. $G_{\text{temp2}} = \emptyset$
11. **For** $1 \leq j \leq$ **length**$(\tilde{y}^{\text{var}}_{\text{ind}})$
12. $\quad \text{maskG} = \left(G_{\text{temp1}}(:, 2) == \tilde{y}^{\text{var}}_{\text{ind}}(j)\right)$
13. $\quad G_{\text{temp2}} := \begin{bmatrix} G_{\text{temp2}} \\ G_{\text{temp1}}(\text{maskG}, :) \end{bmatrix}$
14. **End**
15. $\tilde{u}^{\text{var}}_{\text{ind}} =$ **unique**$(G_{\text{temp2}}(:, 5))$*//Remove repetitions.*
16. $\tilde{u}^{\text{fix}}_{\text{ind}} = \text{uLinList} \setminus \tilde{u}^{\text{var}}_{\text{ind}}$*//Obtain the indices of fixed inputs.*
17. $\tilde{u}^{\text{var}} = \tilde{u}(\tilde{u}^{\text{var}}_{\text{ind}})$*//The variable input vector.*
18. $\tilde{u}^{\text{fix}} = \tilde{u}(\tilde{u}^{\text{fix}}_{\text{ind}})$*//The fixed input vector.*

Chapter 5

Case Studies

5-1 Introduction

This chapter presents the results of two test cases applied to a small, virtual network. In each test case, a train run is disturbed by increasing its running time or dwell time. The algorithm as presented in Chapter 4, is used for simulation and control of the network. Test case 1 is used to show the effect of supplying different estimates at various points in time. Also, the effect the computation time δ_t can have on the solution is demonstrated. The second test case demonstrates how much CPU time can be saved when using a control horizon. The resulting network situation are visualized through the use of so-called place-time diagrams, by which the movement of trains over time through the network can be easily interpreted. This chapter is outlined as follows. The test network is presented in Section 5-2. Section 5-3 presents presents the test cases, of which the results are presented in Section 5-5.

5-2 Test Network

The test network [27] is depicted in Figure 5-1. The network consists of 7 stations and 3 train lines. The lines end at stations 1, 6 and 7, where the trains turn around. Stations 2, 3, 4 and 5 have overtaking possibilities. At station 3, tracks cross and merge.

The lower drawing in Figure 5-1 shows a schematization of the train lines using the network and shows for each line at which station the trains stop. Lines 1 and 2 are operated by intercity trains, whereas line 3 is operated by a local train. Line 1 connects stations 6 and 7, via station 3 and 5. Line 2 runs between stations 1 and 6, while stopping along the route at stations 3 and 5. Line 3 also connects stations, but stops at all stations in between, viz. stations 2, 3, 4 and 5. The timetable according to which the lines operate, is shown in Table 5-1 and Table 5-2. For each train line, the departure time (d.) and arrival time (a.) are given in minutes of every hour.

The movement of trains through the network over time can be visualized by means of a place-time diagram, as depicted in Figure 5-2 and Figure 5-3. Each color represents a different

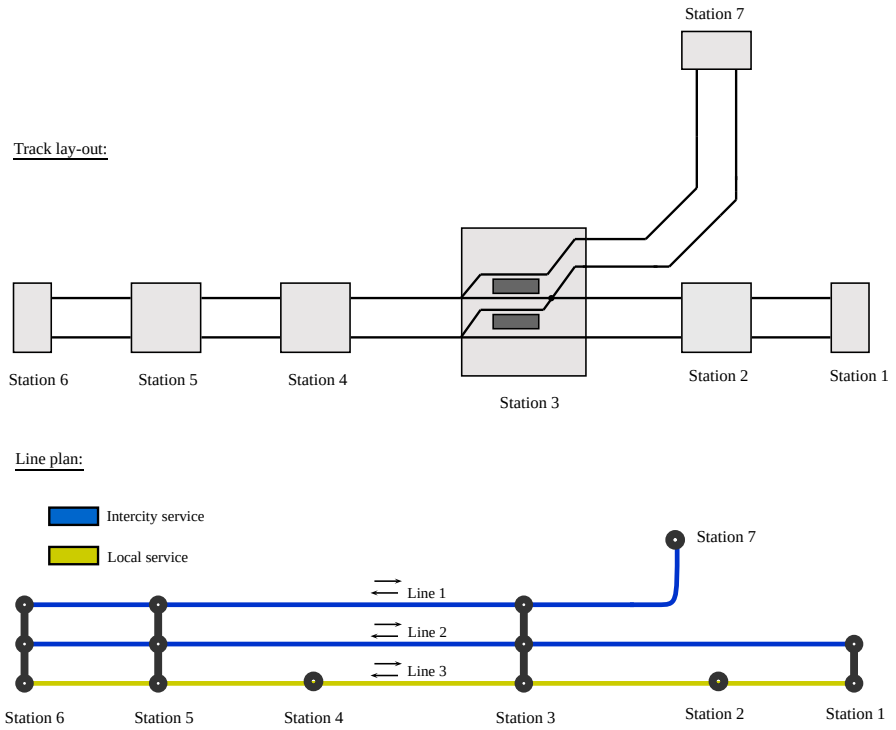

Figure 5-1: The network used for the case studies as described in this chapter. Above: the track layout. Below: the line plan.

		Line 1		Line 2		Line 3	
Station 1	d.			.15	.45	.21	.51
Station 2	a.			|	|	.28	.58
	d.			.20	.50	.29	.59
Station 7	d.	.00	.30	|	|	|	|
Station 3	a.	.11	.41	.26	.56	.36	.06
	d.	.13	.43	.28	.58	.38	.08
Station 4	a.	|	|	|	|	.45	.15
	d.	.18	.48	.33	.03	.51	.21
Station 5	a.	.28	.58	.43	.13	.04	.34
	d.	.30	.00	.45	.15	.05	.35
Station 6	a.	.36	.06	.51	.11	.13	.43

Table 5-1: Schedule for trains running from right to left in Figure 5-1

train line. In this case, line 1 is shown in red, line 2 in blue and line 3 in green. Each plot is accompanied with a number, like '301' or '102'. The first digit denotes the train line number to which the train run is associated. The second two digits are reserved for the cycle number to which the train run belongs. The vertical lines in Figure 5-3 belong to lines using the crossing at station 3. This crossing is modelled as a track with a running time equal to 0 minutes. Note that line 101 using the crossing represents line 1 running from station 6 to 7 and does not belong to train 101 running between station 3 and 6. The same holds for the plots associated to trains 102.

		Line 1		Line 2		Line 3	
Station 6	d.	.16	.46	.01	.31	.09	.39
Station 5	a.	.22	.52	.07	.37	.17	.47
	d.	.24	.54	.09	.39	.18	.48
Station 4	a.	|	|	|	|	.31	.01
	d.	.34	.04	.19	.49	.37	.07
Station 3	a.	.39	.09	.24	.54	.44	.14
	d.	.41	.11	.26	.56	.46	.16
Station 7	d.	.52	.22	|	|	|	|
Station 2	a.			|	|	.53	.23
	d.			.32	.02	.54	.24
Station 1	a.			.38	.08	.01	.31

Table 5-2: Schedule for trains running from left to right in Figure 5-1

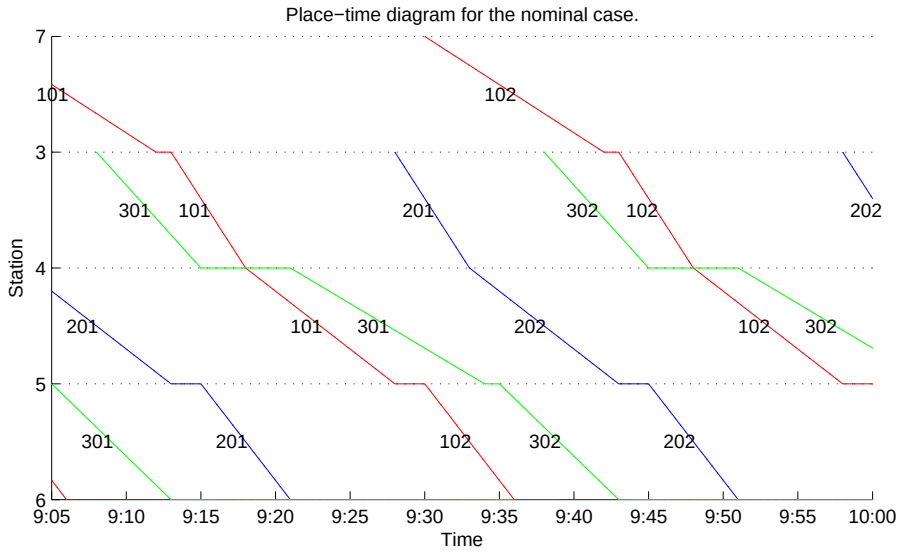

Figure 5-2: Place-time diagram the lines running between stations 7, 3 and 6 in the case of nominal operation.

From the timetable, matrices G and H can be generated, but first the timetable must be converted to the format as given in Table 4-1. Appendix A-1 shows this timetable format for the test network.

The next section presents the to be evaluated delay scenarios.

5-3 Test cases

Using the test network in Figure 5-1, various test cases have been evaluated. In each test case, the network operation is perturbed by changing a running or dwell time. The magnitude of a disturbance has an upper limit. This upper limit is equal to the period of the affected train line. Imposing this limit is motivated by the assumption that dispatching actions other

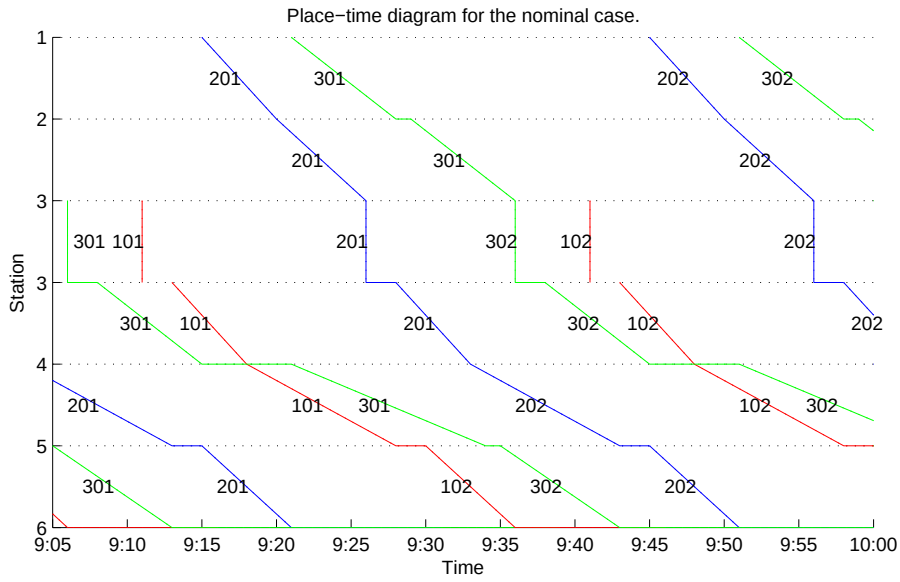

Figure 5-3: Place-time diagram for the lines running between stations 1 and 6 in the case of nominal operation.

than order swaps are more effective in such cases. For example, when a train is delayed so much, that it gets caught up by the next train in the schedule, servicing the same line, then coupling the delayed train to this next train, or just removing the delayed train from the route, is probably preferred.

Recall that the the format in which disturbance information is supplied (see Section 4-4-1), is given by:

$$Z_i = [\tilde{\Theta}_{\text{ind}} \mid \tilde{\Theta}_{\text{est}}(\tilde{\Theta}_{\text{ind}}) \mid t_{\text{est}}],$$

The first entry points to the index of the running time in the parameter vector. The second contains the disturbed parameter value, which equals the nominal value plus the disturbance magnitude. The third entry gives the time of estimation.

Test case 1

In this test case, the effect of the minimum re-schedule time constraint is demonstrated. Two disturbance scenarios are created:

Case 1.1 Delaying the run of line 1 between stations 3 and 4 (train 101)

Case 1.2 Delaying the departure of line 1 from station 5 (train 101).

In the first test, denoted by case 1.1, the running time of train 101 between station 3 and 4 is perturbed. Two estimates are supplied at different times. The estimates are chosen such that they result in different decisions. In this case, the first estimate turned out to be too low,

time	train	estimated disturbance (minutes)
9:15	101, between station 3 and 4	+5
9:21	101, between station 3 and 4	+10

Table 5-3: Data for test case 1.1

time	train	estimated disturbance (minutes)
9:30	101 at station 5	+5
9:33	101 at station 5	+8

Table 5-4: Data for test case 1.2

such that the second estimate, which is assumed to be the actual value of the extra running time, results in an order swap. The test case data is given in Table 5-3.

In the second test, denoted by case 1.2, the dwell time of train 101 at station 5 is perturbed. Again, two estimates are supplied at different times and are again chosen such that they result in different decisions. The test case data is given in Table 5-4.

Test case 2

In this test case, the effect of the control horizon on the computation time is assessed. First, the network is simulated with perturbed parameters without applying control. Then, the network is tested with the receding horizon controller without control horizon. The resulting total delays and average computation times are recorded after each test and compared against the results obtained without applying control. The same tests are carried out with a control horizon of various lengths, ranging from 5 minutes to 30 minutes with 5 minute intervals. The resulting total delays and average computation times are recorded as well.

The following two trains cases are evaluated:

Case 2.1 Delaying the run of line 1 between stations 7 and 3 (train 101)

Case 2.2 Delaying the run of line 3 between stations 1 and 2 (train 301).

The following 5 disturbance magnitudes are applied to each train run:

- +5 min.
- +10 min.
- +15 min.
- +20 min.
- +25 min.

These scenarios are chosen since order swaps are most likely to be effective in reducing the total delay in the network.

5-4 Setup and Settings

The simulations are carried out on a personal computer with the specifications as listed in table Table 5-5.

Operating system	Windows 7 Enterprise SP1, 32-bit.
Processor	Intel® Core ™2 Duo E8500 @ 3.16GHz
RAM	4.00 GB

Table 5-5: PC specifications.

Some parameter values, like the length of the prediction horizon and the value for β, still need to be defined. Their values are given in Table 5-6. The length of the prediction horizon N_p must be large enough to be able to foresee the effects of disturbances and control actions, but must be limited to ensure practical computation times. Furthermore, it does not make sense to look further than three hours ahead in time, as the network situation becomes more and more uncertain the further one looks. Putting more effort in finding the optimal actions is therefore less likely to be paid off. The value for β must be chosen such that all event times within the prediction horizon will always be much smaller than β during the simulation.

Simulation time:	$t_{\text{sim}} = 180$ minutes
Prediction horizon:	$N_p = 6$ cycles
Constant	$\beta = -2000 << -(t_{\text{sim}} + (N_p - 1) \cdot 30)$
Headway time (same dir.)	$\tau_f = 3$ minutes
Headway time (opposite dir.)	$\tau_w = 1$ minute
Max. comp. time	$\delta_t = 2$ minutes
Weighting of $\tilde{y}$	1
Weighting of $\tilde{u}$	$1e^{-5}$

Table 5-6: Simulation parameters.

The value for δ_t is fictive and constant. The value is merely chosen to prove a concept, viz. the working of the receding horizon controller and to the effect of the minimum re-schedule time constraint. Due to the large number of variables in a practical situation, the computation time can be large and will significantly affect the performance of the controller. The actual computation times for the test network were never higher than 1.4 seconds.

5-5 Results

The results of test case 1 and 2 are presented below.

5-5-1 Test case 1.1

Figure 5-4 to Figure 5-6 show the (partly estimated) network situation at various times for test case 1.1. Figure 5-4 shows the situation at 9:00h, when operation is still nominal.

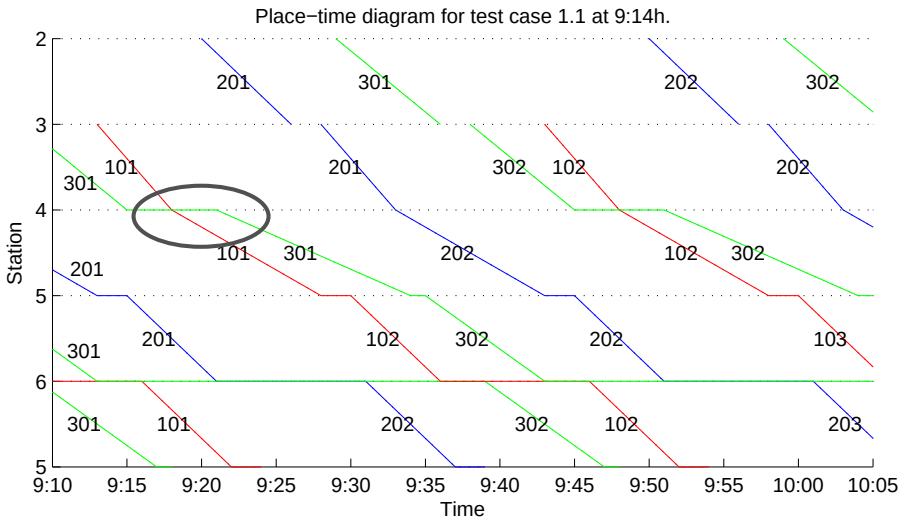

Figure 5-4: Network situation at 9:00h in test case 1.1. All event times later than 9:00h are estimates.

When at 9:15h the first estimate is available, the controller computes the event times and train orders as depicted in Figure 5-5. Note that this advice is available at 9:17h, which is δ_t minutes later than the time the estimate was supplied. No order swaps are advised at this point.

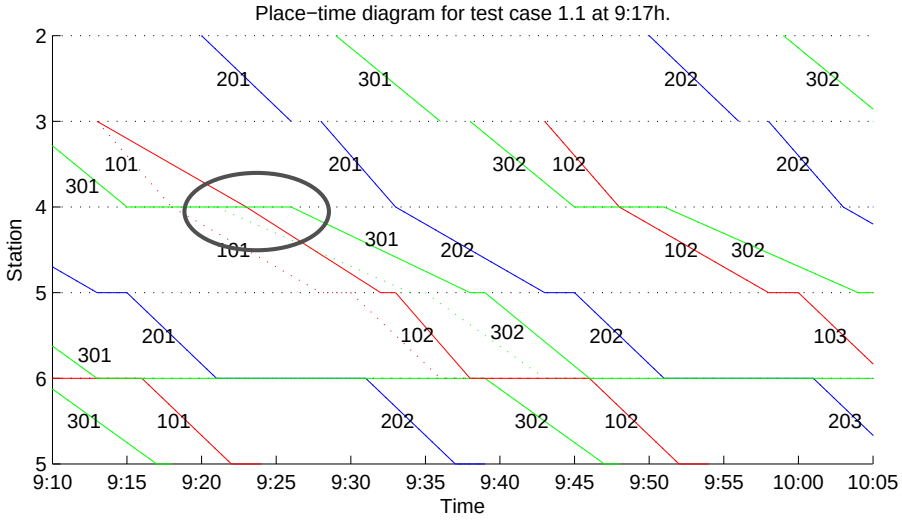

Figure 5-5: Network situation at 9:17h in test case 1.1. All event times later than 9:17h are estimates. No order swap is advised yet.

Then, at 9:21h a new estimate is available and the optimization is carried out. At 9:23h, the optimization is finished and an order swap of train 101 and 301 is advised (Figure 5-6). Note that train 301 is restricted to depart not earlier than 9:23h, while its scheduled time was 9:21h. This is due to the minimum re-schedule time constraint of the receding horizon controller.

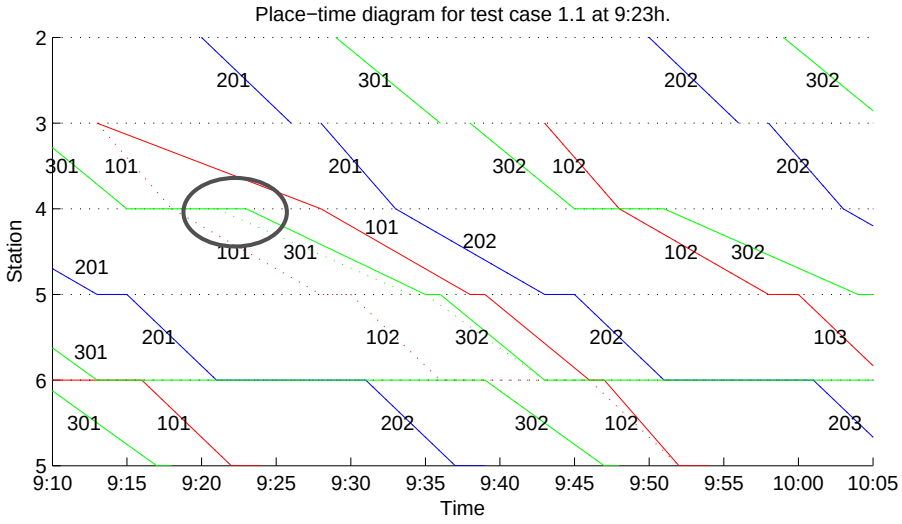

Figure 5-6: Network situation at 9:23h in test case 1.1. All event times later than 9:23h are estimates. The order of train 301 and 101 is changed. However, train 301 cannot depart at its scheduled time of 9:20h.

5-5-2 Test case 1.2

Figure 5-7 to Figure 5-9 show the (partly estimated) network situation at various times for test case 1.2. Figure 5-7 shows the nominal situation.

At time 9:30h, train 102 should have left, but the driver has not shown up yet. It is estimated he will arrive in 5 minutes. Two minutes later, the optimization routine is finished and it turns out it is best to let train 302 wait. However, at 9:33h, the driver still has not shown up and a new estimate is supplied that he will arrive in another 3 minutes. The optimization is restarted and at time 9:35h the controller presents the solution in which train 302 can depart before train 102 without delay. This is depicted in Figure 5-9.

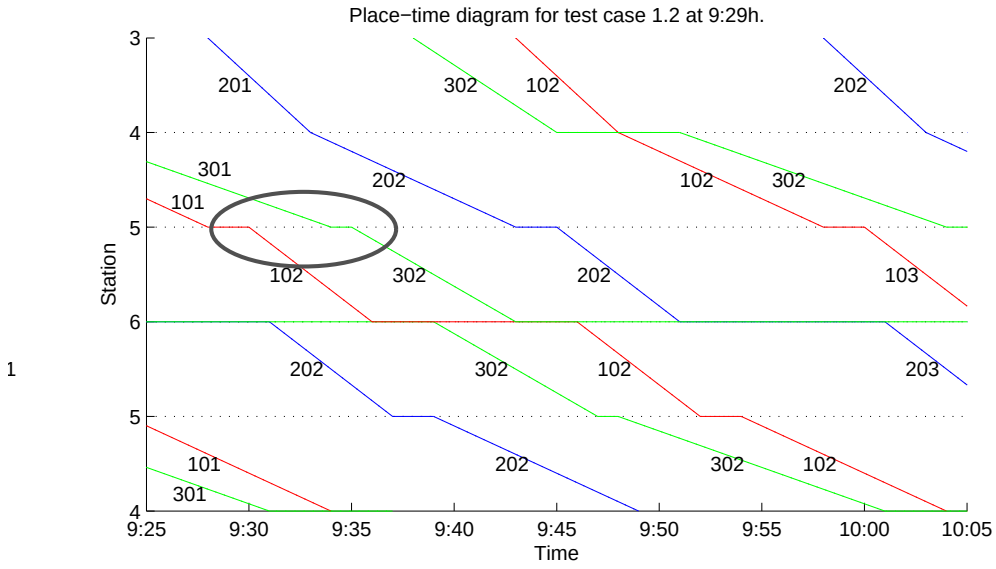

Figure 5-7: Network situation at 9:29h in test case 1.2. All event times later than 9:29h are estimates.

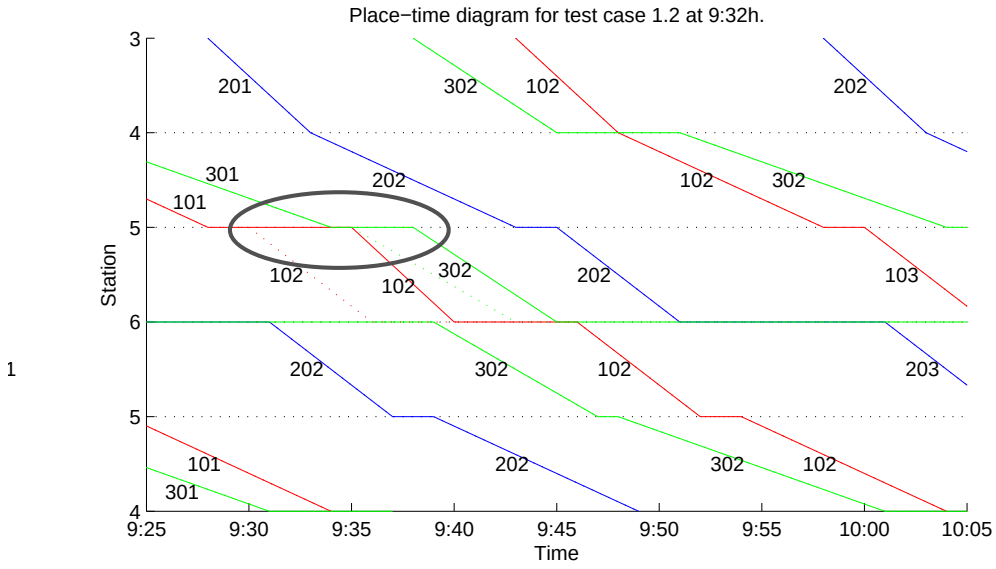

Figure 5-8: Network situation at 9:32h in test case 1.2, after the first optimization. All event times later than 9:32h are estimates.

In case 1.2, no extra delay was introduced with the given computation time and the time at which the estimation was supplied. Suppose the computation time δ_t is not equal to 2 minutes, but 4 minutes. In that case, the situation changes to that as depicted in Figure 5-10. The second estimate came too late for the order of train 302 and 101 to be changed. Note that the second estimate was supplied during the optimization resulting from the first estimate at 9:30h. In that case, the optimization routine is stopped and re-initiated with the current event times and the new parameter estimate.

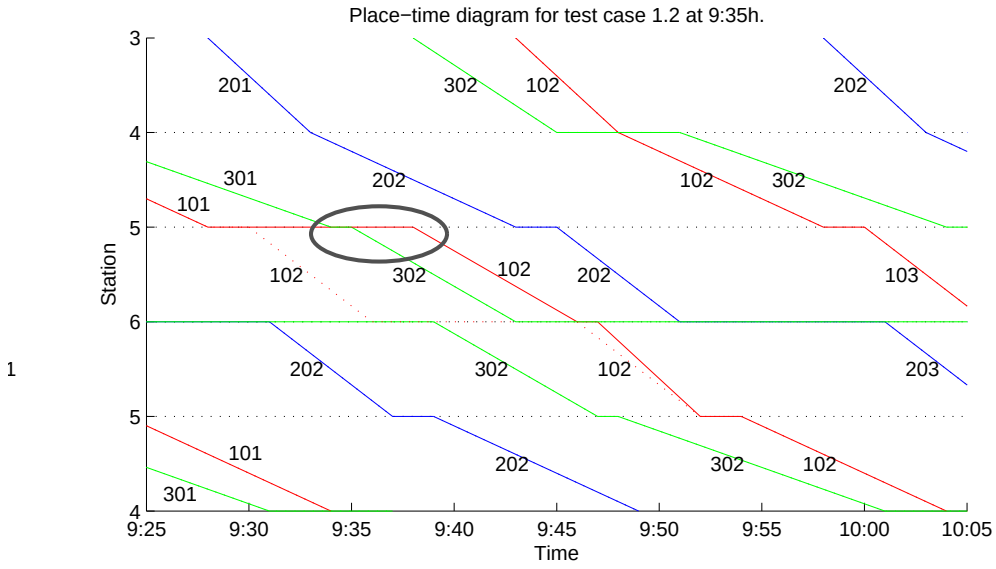

Figure 5-9: Network situation at 9:35h in test case 1.2, after the second optimization. Train 302 is allowed to leave at its scheduled time. All event times later than 9:35h are estimates.

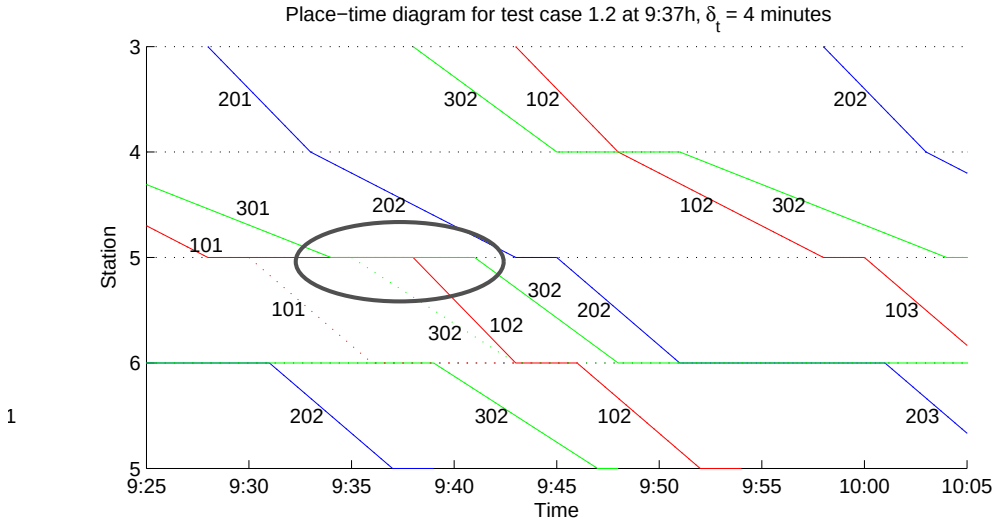

Figure 5-10: Network situation at 9:37h in test case 1.2 with $\delta_t = 4$ minutes. Train 302 is forced to leave with a delay.

5-5-3 Test case 2.1

For each scenario, the total delay resulting from a free run (i.e. without applying control) is compared to the total delay obtained when applying the receding horizon control algorithm without control horizon (full control). Part of this total delay is the sum of delayed departures and arrivals of the affected train. This delay cannot be removed by applying order swaps, but will vanish due to slack times in the timetable. Figure 5-11 and Figure 5-12 show the resulting total delays for both scenarios. Note that the total delay shown in the graphs is the sum of *all* delayed events.

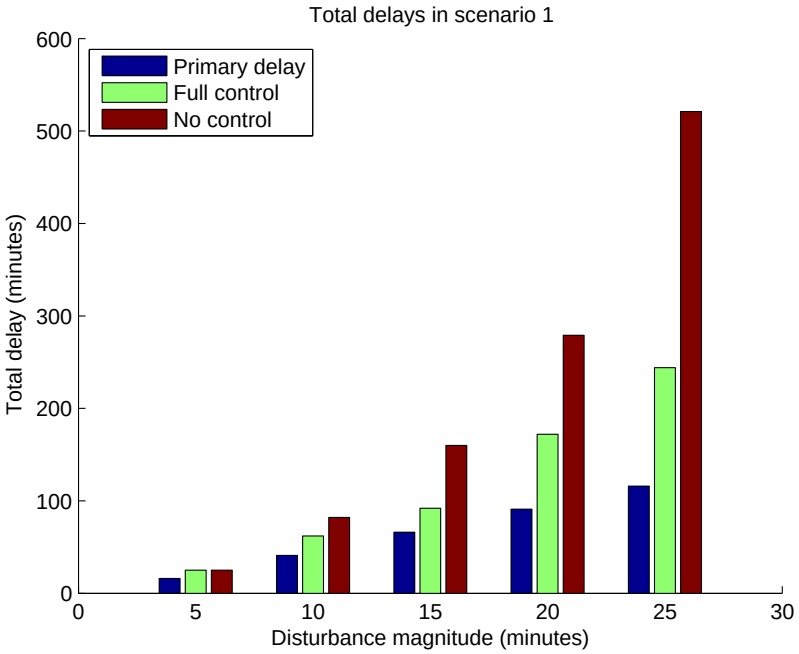

Figure 5-11: Total delay with and without applying control in scenario 1.

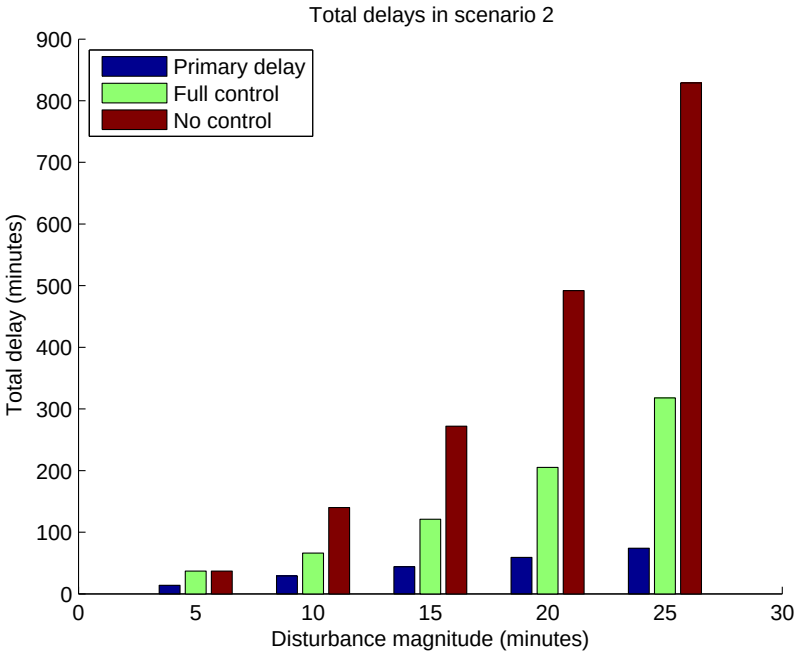

Figure 5-12: Total delay with and without applying control in scenario 2.

In both cases, the delays resulting from a 5 minute disturbance cannot be removed by swapping train orders, but has to die out due to the slack times present in the timetable. As the disturbance magnitude becomes bigger, the order swaps become more effective. The place-

time diagrams for scenario 1 with a source delay of 10 minutes is shown in Figure 5-13. From this figure the order swap between train 301 and 101 at stations 4 and 5 can be seen.

The place-time diagrams for scenario 2 with a source delay of 10 minutes is shown in Figure 5-14. Note the order swap between trains 302 and 102 at the crossing.

5-5-4 Test case 2.2

Each scenario was also simulated with the algorithm using a control horizon with various lengths. From these simulations, the total delay and average computation times were recorded and compared against those resulting from the simulations without the use of a control horizon. The results are shown in Figure 5-15 and Figure 5-16.

In both scenarios, the same amount of total delay is obtained, with a reduction in computation time of up to 36.7%. The reduction does not seem to have a clear relation with the size of the delay. However, the graphs lie relatively close to each other and it appears that a reduction of around 30% can be expected using a control horizon in this network.

5-6 Conclusions

In the test cases as presented in this chapter, the working of the receding horizon controller was demonstrated. It was shown how the algorithm copes with various estimates supplied at different time instants. Also, the effect of the computation time δ_t was demonstrated. The use of a control horizon provides a significant reduction of computation times and can be viewed as a practical heuristic to reduce the computational complexity of finding optimal order swaps.

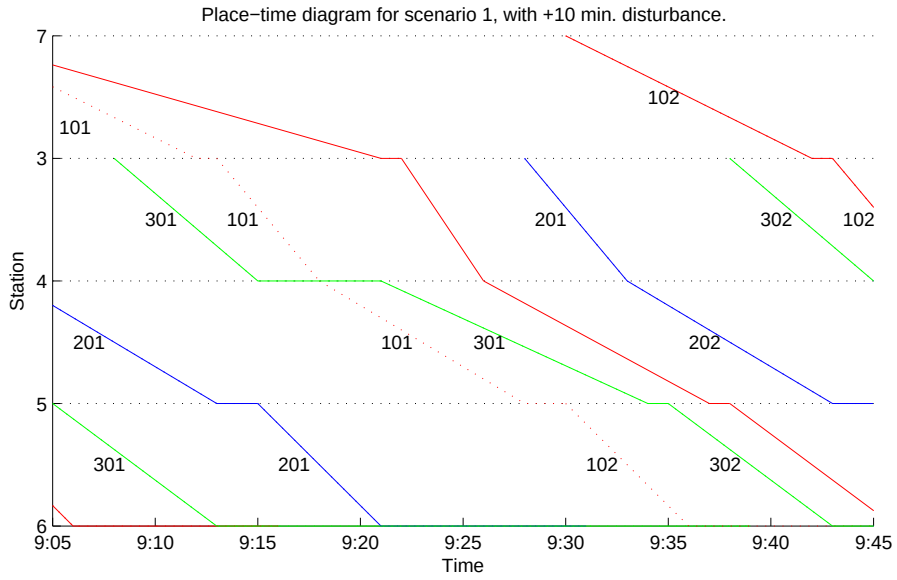

Figure 5-13: Place-time diagram for the lines running between stations 7, 3 and 6 during scenario 1, in which train 101 is delayed 10 minutes. The actual movement is shown with a solid line, whereas the scheduled movement is shown with a dotted line.

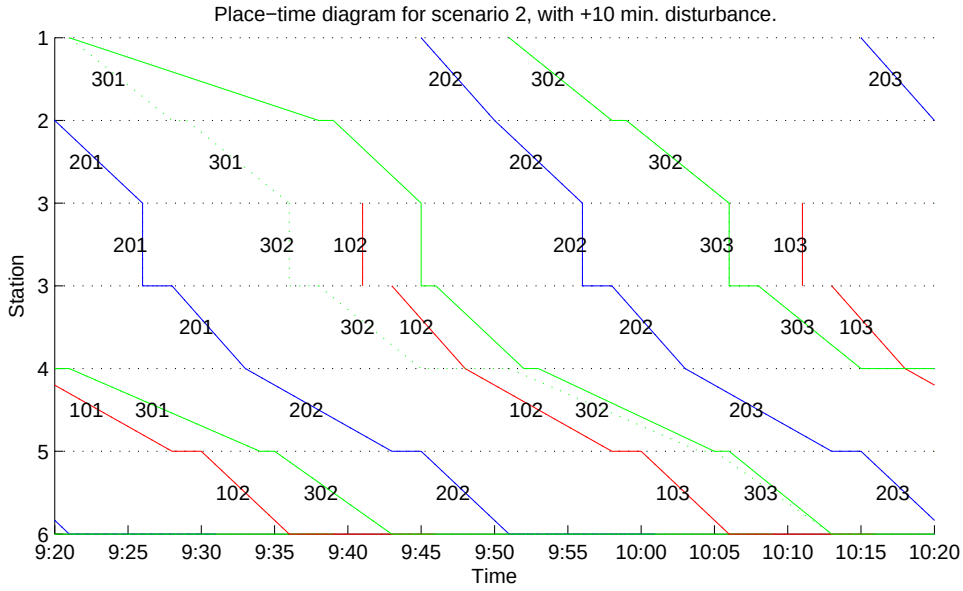

Figure 5-14: Place-time diagram for the lines running between stations 1 and 6 during scenario 2, in which train 301 is delayed 10 minutes.

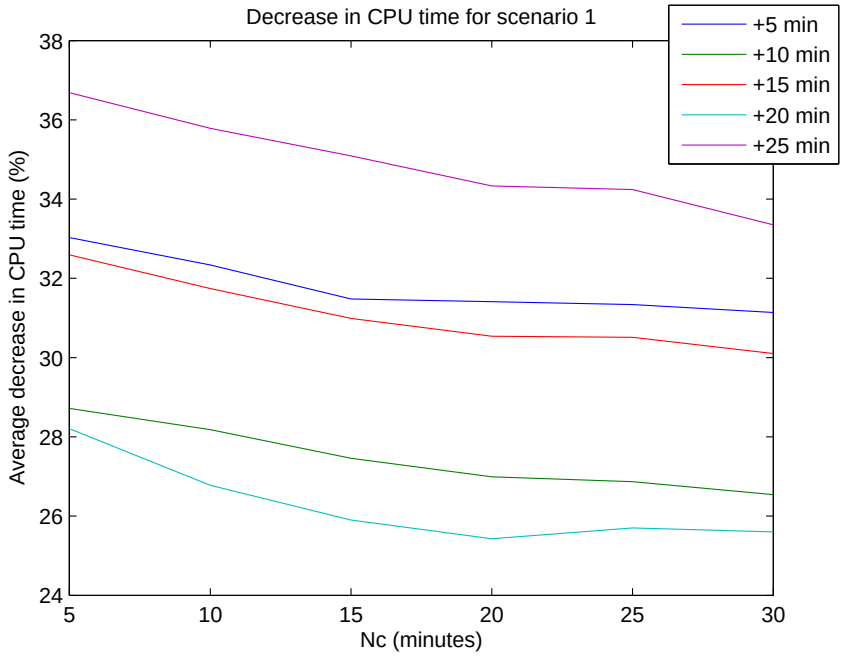

Figure 5-15: Reduction in computation time compared to full control for various lengths of N_c for scenario 1. Each line is associated to a specific disturbance magnitude.

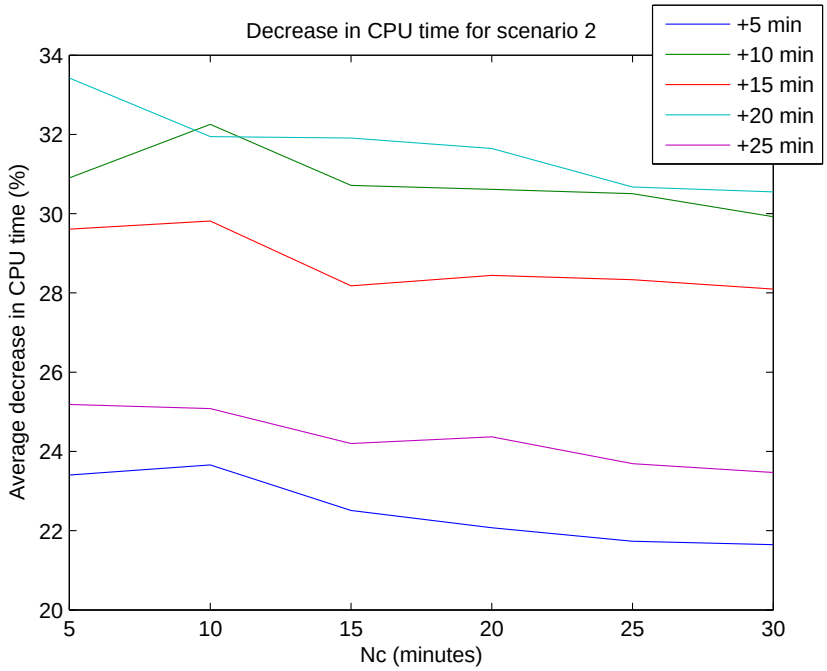

Figure 5-16: Reduction in computation time compared to full control for various lengths of N_c for scenario 2. Each line is associated to a specific disturbance magnitude.

Chapter 6

Conclusions and Recommendations

In this thesis, the development of a receding horizon algorithm for railway networks has been presented. This algorithm was implemented in MathWorks MATLAB and tested with a virtual network to which various disturbances were applied. Two cases were created. In the fist test case, different parameter estimates were supplied to the receding horizon controller at various points in time. This test demonstrated the effect of the minimum re-schedule time constraint. The second test case showed that a control horizon can significantly reduce the computation time needed. The text below presents a discussion of the results and how the various aspects apply in a practical setting. Finally, some recommendations for future work are presented.

6-1 Conclusions

The structuring of the max-plus model as presented in Chapter 2 and Chapter 3 provides a convenient notation of complex railway network models. The use of max-plus algebra allows for the application of max-plus system theory to analyze or uncover interesting properties of railway networks. Although the model only contains running, dwell and headway constraints, it can be easily extended to also contain e.g. connection constraints. Connection constraints link the departure of on physical train to the arrival of another physical train, such that passengers are given the time to transfer from one train to the other. By using a binary input associated to these constraints, connections can be broken in a very similar way the headway relations are reversed. A less practical aspect of the model, is that it is assumed that each cycle consists of the same number of events. Usually, during rush-hours, more trains are driving, so more events occur per cycle. This also means that the model of the railway network changes. A solution could be to use a model containing all events that may occur during one cycle. Then, during rush-hour, only the events associated to this mode of network operation are considered, while all other events can be fixed. Switching to another mode of operation can be done by changing which events are fixed and which are variable. The algorithm presented in this thesis can then also be applied to railway networks using different timetables during different parts of the day.

The developed receding horizon control algorithm allows for optimal control of a railway network, with the possibility to supply varying parameter estimations. The optimization problem is done over only a set of variable event times and inputs and further reduction of the MILP can be achieved by just changing the set of variable event times and inputs. The algorithm offers a general framework which allows for convenient further development. Furthermore, the algorithm can work with practical constraints such as the effect of a minimum computation time, by employing a minimum re-schedule time constraint. The first test case showed the effect of this minimum re-schedule time constraint, which was introduced such that delayed events could not be re-scheduled to a point in time *before* the time the optimization was finished. The decision of the algorithm would then not be based on the actual network situation any longer. The optimal event times and train orders are not known until the end of the optimization, thus the algorithm has to take into account its own computation time. It seems that the computation time is the weakest link in the chain. After all, when the computation time δ_t is small, more decisions can be evaluated which improve the solution. However, now we are asking for more freedom for less computation time, which is contradictive. We should therefore find a method to determine which decision variables are not needed to find the global optimum, i.e. the best decision from a network point of view. In the test cases, the computation time was assumed to be constant and known beforehand. In practice, the value for δ_t could then be based on a worst case scenario, in which the computation time needed is largest. However, this worst case computation time should not be based on an unlikely scenario, as then most of the times the algorithm can find better solutions with smaller values of δ_t. If a scenario occurs in which more computation time is needed than expected, just implementing the best solution obtained so far is probably the most practical solution. Another approach is to make δ_t dynamic, i.e. changing per scenario. As the disturbances get bigger and more trains are involved, it is more likely that more order swaps will occur. During testing, it was noticed that the average computation time needed was bigger in those cases. In that light, it might be not bad that some events get a little extra delay due to computation time, as the overall reduction in delay might be very good. Furthermore, what is done when an event is delayed during optimization, which was not delayed when the optimization started? Does the optimization routine need to be re-started every time that happens? If the decisions resulting from the disturbance of two separate events are independent *in the optimum*, these problems can be decoupled. In that case, two separate optimizations can be carried out. This requires knowledge on how these disturbances interact in the optimum. It is a very realistic aspect of the algorithm to include the minimum re-schedule time and provides a good framework to study the use of more sophisticated methods, like assigning dynamic values to δ_t.

Besides identifying variables which will not change in the optimum, the application of a control horizon is a good method to reduce the computation time. This control horizon should then be as small as possible. However, how does one know beforehand how small this is? One method could be to first simulate the network with the perturbed parameter vector to see how far in the future events are affected by the disturbance. This would than be an upper boundary for the length of the control horizon. Although from the results of test case 2 it seemed that the control horizon could be as short as one wanted while still obtained the global optimum, this is not the case. The algorithm using a control horizon bases its decisions without 'knowing' about the possibilities of order swaps beyond this control horizon. Therefore, it could 'miss' a better alternative. However, it could be a practical measure to keep the computation time short. What is not implemented in the algorithm, is a time-based prediction horizon. Instead,

an event-based prediction horizon was used. This event-based prediction horizon is defined over an integer number of cycles and recedes as soon all the event times of the earliest cycle are known. This, however, leaves very little room for tweaking, since adding a cycle to the prediction horizon means adding n events with all the inputs associated to those events. The number of inputs per track grows exponentially for each train added to that track. Just like the control horizon, the prediction horizon could be defined over a time span. This allows for more freedom in calibrating the prediction horizon and thus the computation time. The algorithm presented in this thesis provides easy implementation of such a prediction horizon. Based on the results obtained and the experience gained, the following recommendations are made regarding the further development of the algorithm.

6-2 Recommendations For Future Work

The text below proposes some extensions to the algorithm as presented in this thesis as well as ideas for future endeavours.

6-2-1 Time-based prediction horizon

Just as was done with the control horizon N_c, the prediction horizon N_p could be defined over a fixed time span. The system matrices are generated beforehand describing events within N_s cycles. Every time step, the prediction horizon includes events N_p time steps in the future. All events that lie outside this time span are included in the set $\tilde{y}^{\text{fix}}$. Only Algorithm 8 needs to be rewritten into to Algorithm 18.

The size of N_s should be chosen such that, in the case of a severe disturbance, at any time during the re-scheduling procedure, the prediction horizon can still recede to include new events. A rule of thumb would then be:

$$N_s = \lceil \frac{N_p}{T} \rceil + 2 \tag{6-1}$$

Instead of defining the prediction horizon over a time span, it could also be defined over a fixed number of events.

6-2-2 Dynamic control/prediction horizon length

By first simulating the network with the perturbed parameter vector while fixing the input vector at its previous optimal values, the extend of the disturbance can be assessed. From this information, the maximum length for the control and/or prediction horizon can be set.

6-2-3 Control horizon over successive events

The control horizon could be defined over only the *affected* events within some time frame. This reduces the search space by excluding control variables which would never improve the objective function [10]. All events not in the successive events list can be fixed, reducing the size of the MILP, thus decreasing the computation time.

Algorithm 18 Separation of fixed and variable states. All variable states lie within a N_p minute time frame.

Syntax:

$[y^{\text{mask}}, \tilde{y}^{\text{fix}}_{\text{ind}}, \tilde{y}^{\text{var}}_{\text{ind}}, \tilde{y}^{\text{fix}}, \tilde{y}^{\text{var}}] = \text{fixStates}(\tilde{y}, t, n, N_p, N_s)$

Output:

y^{mask} *//A binary mask with 1's on the places corresponding to fixed events.*

$\tilde{y}^{\text{fix}}_{\text{ind}}$ *//Indices of fixed events.*

$\tilde{y}^{\text{var}}_{\text{ind}}$ *//Indices of variable events.*

$\tilde{y}^{\text{fix}}$ *//The fixed event times.*

$\tilde{y}^{\text{var}}$ *//The variable event times.*

Input:

$\tilde{y}$ *//The event time vector.*

t *//The current time.*

N_s *//The number of cycles for which the model is generated*

n, N_p *//The number of trains per cycle and the length of the prediction horizon.*

1. $\text{yLinList} = [1, 2, \ldots, 2 \cdot n \cdot N_s]^T$
2. $y^{\text{mask}} = (\tilde{y} \leq t) \& (\tilde{y} > t + N_p)$
3. $\tilde{y}^{\text{fix}}_{\text{ind}} = \text{yLinList}(y^{\text{mask}})$
4. $\tilde{y}^{\text{var}}_{\text{ind}} = \text{yLinList}(!y^{\text{mask}})$
5. $\tilde{y}^{\text{fix}} = \tilde{y}(\tilde{y}^{\text{fix}}_{\text{ind}})$
6. $\tilde{y}^{\text{var}} = \tilde{y}(\tilde{y}^{\text{var}}_{\text{ind}})$

6-2-4 Effect of parameter variation

Determining how much variation in disturbance estimates can be tolerated without it affecting the decisions is an interesting topic to investigate. If a change in parameter value does not affect the decision, this saves an optimization run. If no optimization run is needed, no delay due to the optimization is introduced.

Appendix A

Appendix

A-1 Timetable for the test network

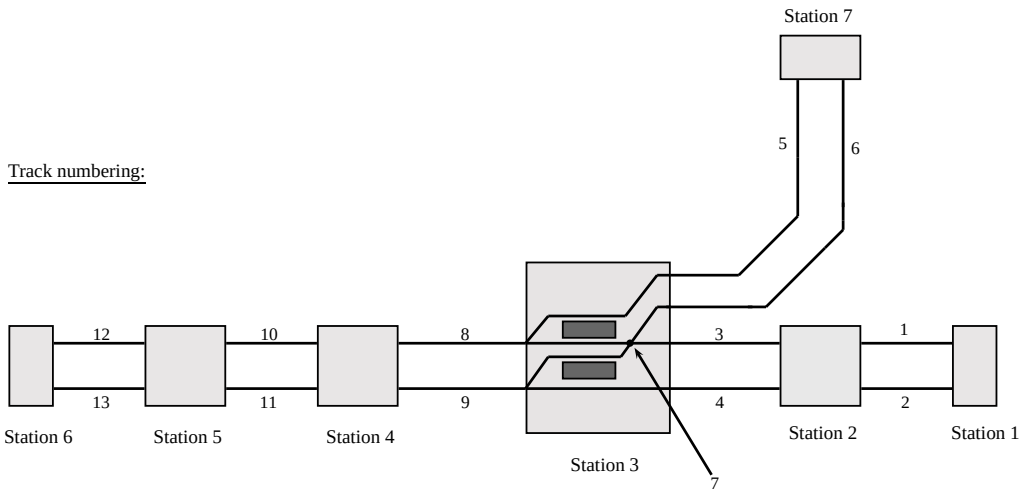

Figure A-1: Track numbering of the test network.

The timetable in the format as presented in Section 4-2-1, is shown in Table A-1. Recall that a crossing is represented by a virtual track, as explained in Section 2-1. This virtual track track number 7 in Figure A-1.

train	line	track	dir.	dep. time	run. time	prev.train	cycle	dwell time	arr. time
1	1	5	0	0	11	9	-1	1	12
2	1	8	0	13	4	1	0	1	18
3	1	10	0	18	9	2	0	0	28
4	1	12	0	0	5	3	-1	1	6
5	1	13	0	16	5	4	0	1	22
6	1	11	0	24	9	5	0	1	34
7	1	9	0	4	4	6	-1	0	9
8	1	7	0	11	0	7	0	1	11
9	1	6	0	11	11	8	0	0	23
10	2	1	0	15	4	20	0	1	20
11	2	3	0	20	5	10	0	0	26
12	2	7	-1	26	0	11	0	0	26
13	2	8	0	28	4	12	0	1	33
14	2	10	0	3	9	13	-1	0	13
15	2	12	0	15	5	14	0	1	21
16	2	13	0	1	5	15	-1	1	7
17	2	11	0	9	9	16	0	1	19
18	2	9	0	19	4	17	0	0	24
19	2	4	0	26	5	18	0	1	32
20	2	2	0	2	4	19	-1	0	7
21	3	1	0	21	6	31	-1	1	28
22	3	3	0	29	6	21	0	1	36
23	3	7	-1	6	0	22	-1	0	6
24	3	8	0	8	6	23	0	1	15
25	3	10	0	21	12	24	0	1	34
26	3	12	0	5	7	25	-1	1	13
27	3	13	0	9	7	26	-1	1	17
28	3	11	0	18	12	27	0	1	31
29	3	9	0	7	6	28	-1	1	14
30	3	4	0	16	6	29	0	1	23
31	3	2	0	24	6	30	0	1	31

Table A-1: The timetable for the test network.

Bibliography

[1] M. J. Vromans, R. Dekker, and L. Kroon, "Reliability and heterogeneity of railway services," *European Journal of Operations Research*, vol. 172, pp. 647–665, July 2006.

[2] D. Huisman, L. G. Kroon, R. M. Lentink, and M. J. Vromans, "Operations research in passenger railway tranportation," *ERIM Report series: Research in management (ref: ERS-2005-o23-LIS (Logistics))*, April 2005.

[3] J. Jespersen-Groth, D. Potthoff, J. Clausen, D. Huisman, L. Kroon, G. Maróti, and M. Nyhave Nielsen, "Disruption management in passenger railway transportation," *Econometric Institute Report EI2007-05*, 2010.

[4] P.-J. Fioole, L. Kroon, G. Maróti, and A. Schrijver, "A rolling stock circulation model for combining and splitting of passenger trains," *European Journal of Operational Research*, vol. 174, pp. 1281–1297, October 2006.

[5] G. Budai, G. Maróti, R. Dekker, D. Huisman, and L. Kroon, "Rescheduling in passenger railways: The rolling stock rebalacing problem," *Journal of Scheduling, Springerlink*, vol. 13, pp. 281–297, September 2009.

[6] M. Gatto, R. Jacob, L. Peeters, and A. Schöbel, "The computational complexity of delay management," *Lecture Notes in Computer Science*, vol. 3787, pp. 227–238, June 2005.

[7] A. Schöbel, "Integer programming approaches for solving the delay management problem," *Lecture Notes in Computer Science: Algorithmic Methods for Railway Optimisation*, vol. 4359, pp. 145–170, September 2004.

[8] M. Gatto, B. Glaus, R. Jacob, L. Peeters, and P. Wildmayer, "Railway delay management: Exploring its algorithmic complexity," *Lecture Notes in Computer Science: Algorithmic Theory - SWAT 2004*, vol. 3111, pp. 199–211, July 2004.

[9] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra, "Delay management problem: Complexity results and robust algorithms," *Lecture Notes in Computer Science: Combinatorial Optimization and Applications, Proceedings*, vol. 5165, pp. 458–468, August 2008.

[10] M. Schachtebeck and A. Schöbel, "To wait or not to wait - and who goes first? delay management with priority decisions," *Transportation Science*, vol. 44, pp. 307–321, August 2010.

[11] A. D'Ariano, M. Pranzo, and I. A. Hansen, "Conflict resolution and train speed coordination for solving real-time timetable perturbations," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, pp. 208–222, June 2007.

[12] A. D'Ariano, F. Corman, D. Pacciarelli, and M. Pranzo, "Reordering and local rerouting strategies to manage train traffic in real time," *Transportation Science*, vol. 42, pp. 405–419, November 2008.

[13] A. W. Andersson, B. Sandblad, P. Hellstrom, I. Frej, and A. Gideon, "A systems analysis approach to modelling train traffic control," *World Congr. Railway Res., Florence, Italy*, pp. 673–679, 1997.

[14] L. Chen, F. Schmid, I. M. Dasigi, B. Ning, C. Roberts, and T. Tang, "Real-time train rescheduling in junction areas," *Proceedings of the Institution of Mechanical Engineers Part F-Journal of Rail and Rapid Transit*, vol. 224, no. F6, pp. 547–557, 2010.

[15] M. Dorfman and J. Medanic, "Scheduling trains on a railway network using a discrete event model of railway traffic," *Transportation Research Part B: Methodological*, vol. 38, pp. 81–98, January 2004.

[16] J. Törnquist, "Computer-based decision support for railway traffic scheduling and dispatching: A review of models and algorithms," in *5th Workshop on Algorithmic Methods and Models for Optimization of Railways* (L. G. Kroon and R. H. Möhring, eds.), (Dagstuhl, Germany), Internationales Begegnungs- und Forschungszentrum f"ur Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[17] B. Adenso-Díaz, M. Olivia González, and P. González-Torre, "On-line timetabel rescheduling in regional train services," *Transportation Research Part B-Methodological*, vol. 33, pp. 387–398, August 1999.

[18] A. Mascis and D. Pacciarelli, "Job-shop scheduling with blocking and no-wait constraints," *European Journal of Operational Research*, vol. 143, pp. 498–517, December 2002.

[19] F. Corman, A. D'Ariano, D. Pacciarelli, and M.Pranzo, "A tabu search algorithm for rerouting trains during rail operations," *European Journal of Operational Research*, vol. 44, pp. 175–192, January 2010.

[20] J. Törnquist and J. A. Persson, "N-tracked railway traffic re-scheduling during disturbances," *Transportation Research Part B*, vol. 41, pp. 342–362, March 2007.

[21] J. Krasemann, "Greedy algorithm for railway traffic re-scheduling during disturbances: a swedish case," *IET Intelligent Transport Systems*, vol. 4, pp. 375–386, December 2010.

[22] T. van den Boom, N. Weiss, W. Leune, R. Goverde, and B. De Schutter, "A permutation-based algorithm to optimally reschedule trains in a railway traffic network," in *Proceedings of the 18th IFAC World Congress*, (Milan, Italy), pp. 9537–9542, Aug.–Sept. 2011.

[23] R. M. Goverde, "A delay propagation algorithm for large-scale railway traffic networks," *Transportation Research Part C: Emerging Technologies*, vol. 18, pp. 269–287, June 2010.

[24] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[25] B. Heidergott, G. J. Olsder, and J. van der Woude, *Max Plus at Work*. Princeton University Press, 2004.

[26] M. Kvasnica, P. Grieder, and M. Baotić, "Multi-Parametric Toolbox (MPT)," 2004.

[27] D. van der Meer, "Modelling railway dispatching actions in switching max-plus linear systems," Master's thesis, Delft University of Technology.