Diss. ETH No. 19826

On the Problem of Sorting Railway Freight Cars

An Algorithmic Perspective

A thesis submitted to

ETH Zurich

for the degree of
Doctor of Sciences (Dr. sc. ETH Zurich)

Jens H. Maue
Dipl. Inform. (Saarland University)
born 4th July 1978 in St.Wendel, Germany

accepted on the recommendation of

Prof. Dr. Peter Widmayer, ETH Zurich
Prof. Dr. Marco Lübbecke, RWTH Aachen
Dr. Matúš Mihalák, ETH Zurich

2011

Abstract

In this thesis the algorithmic foundations of a particular sorting problem from railway optimization are studied. The addressed problem is called *train classification*, and it refers to the fundamental procedure of rearranging the cars of several trains into other compositions of different orders comprising new trains. The train classification methods applied today present rather conservative approaches, and there is a lot of room for systematic improvement by applying optimization methods.

The sorting processes are performed according to plans prepared in advance called *classification schedules*. They are conducted in specific railway facilities called *classification yards*. Without expensive redesigns of existing classification yards, the dwell time of railway cars can be reduced by accelerating the core classification process itself. To this aim, the combinatorial structure of the sorting processes are studied in this thesis in order to provide algorithmic solutions for the abstract problems derived from the practical setting with formal proofs of their efficiency. Conversely, the gained insights are applied to real-world problem instances to show that the theoretical approaches work in practice and improve on the methods applied today.

First of all, a novel encoding of classification schedules is presented. This does not only yield an efficient representation of train classification schedules, but it also allows characterizing feasible classification schedules and is applied to derive an algorithm for computing optimal classification schedules for the core version of the sorting problem. Several practical problem settings then yield further restrictions on a feasible schedule, which are shown to translate to formal constraints in the above mentioned representation. Successful solution approaches are then developed for various infrastructural as well as operational constraints. The former particularly cover different dimensions of limited track space, the latter include train departures and track space varying over time. The approaches comprise efficient exact and approximation algorithms and also integer programming models that allow integrating several such constraints simultaneously. Then, another problem variant deals with the robustness aspect of uncertain input, which here corresponds to disruptions in the railway network resulting in trains arriv-

ing delayed at the classification yard. A model is introduced in which a prepared schedule that became infeasible by an arising disruption is adjusted by inserting some additional steps into the sorting process. Without any assumptions on the amount of disruption, this robust problem variant is shown to be $\mathcal{NP}$-hard. Still, a generic method is provided for the general setting, from which an efficient algorithm for a realistic model of disruption is derived. The developed theoretical approaches are compared to each other and evaluated experimentally using various synthetically derived and real-world traffic instances to show that the achieved insights and solutions are relevant in practice.

Keywords: railway optimization, algorithms, combinatorial optimization, sorting physical objects, shunting of rolling stock, train classification

Zusammenfassung

Diese Dissertation untersucht die algorithmischen Grundlagen eines Sortierproblems aus dem Bereich der Eisenbahnoptimierung. Bei dem betrachteten Optimierungsproblem geht es um das Rangieren von Waggons im Bahngüterverkehr, was einen fundamentalen Arbeitsvorgang im Bahnbetrieb darstellt, bei dem die Wagen mehrerer Güterzüge umgruppiert werden, um neue Güterzüge zu bilden. Bei den heute verwendeten Rangiermethoden handelt es sich um äußerst vorsichtige Ansätze, was viele Möglichkeiten zur Verbesserung durch systematische Optimierungsverfahren birgt.

Die Sortiervorgänge werden anhand zuvor festgelegter Ablaufpläne, sogenannter *Rangierpläne*, durchgeführt und finden in *Rangierbahnhöfen* statt, die exklusiv für diese Rangierverfahren geplant und gebaut wurden. Will man teure und zeitraubende Umbauten bestehender Rangierbahnhöfe vermeiden, lässt sich die Verweildauer von Güterwagen in Rangierbahnhöfen durch eine Beschleunigung des angewandten Rangierverfahrens reduzieren. Zu diesem Zweck werden in dieser Dissertation die Sortierverfahren im Hinblick auf ihre kombinatorische Struktur analysiert. Zur Lösung der abgeleiteten abstrakten Problemstellung werden anschließend Algorithmen entwickelt und ihre Korrektheit und Effizienz formal bewiesen. Die daraus gewonnenen Erkenntnisse werden schließlich auf praktische Probleminstanzen angewandt, um zu zeigen, dass die entwickelten theoretischen Lösungswege in der Praxis funktionieren und die heutigen Rangierverfahren tatsächlich verbessern.

Zu Beginn wird eine neuartige Kodierung von Rangierplänen eingeführt, was nicht nur eine effiziente Darstellung von Rangierplänen sondern auch eine Beschreibung ihrer Gültigkeit erlaubt. Mit Hilfe dieser Repräsentation wird dann ein Algorithmus zur Berechnung optimaler Rangierpläne für die Grundversion des Sortierproblems entwickelt. Verschiedene Gegebenheiten innerhalb des praktischen Problems führen zu zusätzlichen Anforderungen an einen gültigen Rangierplan. Es wird gezeigt, wie sich diese Anforderungen in zusätzliche formale Randbedingungen in der oben genannten Darstellung übertragen lassen. Anschließend werden erfolgreiche Lösungswege für diverse Nebenbedingungen bezüglich der Infrastruktur und des Betriebs

des Rangierbahnhofs entwickelt. Diese umfassen insbesondere verschiedene Beschränkungen des Gleisvolumens, eingeschränkte Abfahrtszeiten fertiggestellter Züge oder zeitabhängige Verfügbarkeiten von Gleisen. Bei den Lösungsmethoden handelt es sich um effiziente exakte und Approximationsalgorithmen sowie Modellierungen ganzzahliger Programme, in denen sich mehrere solcher Nebenbedingungen gleichzeitig integrieren lassen. Eine robuste Problemvariante beschäftigt sich mit Unsicherheiten in der Eingabe, was bei diesem Problem Störungen im Eisenbahnnetz entspricht, welche wiederum verspätete Ankünfte von Güterzügen am Rangierbahnhof nach sich ziehen. Es wird ein Modell eingeführt, bei dem ein vorgefertigter Rangierplan – wenn er durch eine auftretende Verspätung ungültig geworden ist – korrigiert werden kann indem zusätzliche Rangierschritte in den Vorgang eingefügt werden. Für den Fall, dass keine Annahmen zum größtmöglichen Umfang auftretender Verspätungen gemacht werden, wird gezeigt, dass es sich bei der robusten Problemvariante um ein $\mathcal{NP}$-schweres Problem handelt. Gleichwohl wird eine generische Methode für diese allgemeine Situation aufgestellt, von der wiederum ein effizienter Algorithmus für ein realistisches Verspätungsmodell abgeleitet wird. Die entwickelten theoretischen Verfahren werden miteinander verglichen und experimentell unter Verwendung diverser synthetischer und echter Verkehrsdaten evaluiert. Die Experimente zeigen, dass die gewonnenen Einsichten und entwickelten Lösungsverfahren von hoher praktischer Relevanz sind.

Schlagworte: Eisenbahnoptimierung, Algorithmen, Kombinatorische Optimierung, Sortieren, Rangieren von Güterwagen, Bahngüterverkehr

Acknowledgements

First of all, I would like to express my gratitude to my thesis supervisor Peter Widmayer for all his valuable help and encouragement over the years. I have enjoyed countless inspiring discussions with him and have always appreciated his helpful advice. I am also grateful for the good atmosphere in our offices and his support in general. Finally, I would like to thank him for his challenging and interesting questions and for putting me at ease during my PhD examination.

I am also very grateful to Marco Lübbecke for his readiness to examine this thesis. I appreciate his swift and thorough reading of my dissertation with all the attendant paperwork, the pleasant atmosphere during my PhD examination, and especially his thoughtful questions and remarks.

Matúš Mihalák has also earned my sincere thanks. To begin with, I am thankful to him for his willingness to act as my third examiner, for the interesting discussion during my PhD examination, and for all his advice and feedback beforehand. Furthermore, I am grateful for his consistently positive attitude, which contributes so much to the good atmosphere in the institute.

I would also like to say thank you to all the other current and former members of our research group: Tomas Hruz, for his continuous IT support; Marianna Berger, for introducing free coffee; Holger Flier, for the fun at ARRIVAL workshops; Anka Zych, for her contribution to a better gender distribution in the computer science department; Rasto Sramek, for hiking; Andreas Feldmann, for playing Skat and even knowing a third player; Marcel Schöngens, for repeatedly taking care of my pot plants; Yann Disser, for the discussions about Hessian and football; Beat Gfeller, for his advice on the formalities necessary to submit my thesis; Elias Vicari, for letting me occasionally win some Töggeli matches; Marc Nunkesser, for the great parties on his roof terrace; Davide Biló, for bringing some Italian flair into our group; Riko Jacob, for interesting discussions; Leon Peeters, for his catching optimism; Shantanu Das, for throwing a birthday party together; Jörg Derungs, for consequently exposing me to Swiss German; Barbara Heller, for the constant supply of cake; Michael Gatto, for his support as head assistant when I gave my first tutorial class.

As a research assistant, I had the chance to give several tutorials for other groups in our department. For the excellent organization of exercise sessions and exams, I am grateful to Lukas Fässler, Markus Dahinden, and Barbara Scheuner; to Johannes Lengler, Torsten Mütze, and Luca Gugelmann; and to Stefan Wolf and Esther Hänggi.

Conversely, as a PhD student, I had the opportunity to attend interesting courses offered by other research groups. I would like to thank the following for the worthwhile lectures and tutorials: Angelika Steger, Reto Spöhel, and Henning Thomas; Dan Hefetz, Uli Wagner, and Heidi Gebauer; and Juraj Hromkovic and Giovanni Serafini.

I am very grateful to all the as-yet-unmentioned members of the CADMO groups lead by Angelika Steger and Emo Welzl for being friendly and inspiring neighbors.

Moreover, I am thankful for the fruitful collaboration I have enjoyed with all the coauthors from outside our research group. I would like to express my sincere gratitude to Alain Hauser, Christina Büsing, Peter Márton, and Markus Bohlin for their commitment and positive attitude which made our work together a great success.

I had the further pleasure of meeting many interesting people within the ARRIVAL project. I would like to thank everybody involved in this fruitful project, notably Christos Zaroliagis, Rolf Möhring, Ravindra Ahuja, Christian Liebchen, and Gabriele Di Stefano.

I would like to thank Stephan Leber from SBB Infrastruktur for his feedback and supply of traffic data, and the hospitable staff of Rangierbahnhof Limmattal for giving an insight into the operation of their yard.

Last but not least, thank you very much to all my friends and family for their great support during my PhD.

Contents

Preface

In this thesis the algorithmic foundations of *train classification* are studied. Train classification is a special sorting problem originating from the field of railways. It refers to the process of splitting up freight trains into their single cars and reassembling them to form new trains, a process which is essential for freight transportation. The train classification process takes place in special installations of railway tracks and switches called *classification yards*. There are *inbound trains* arriving at the yard that carry the cars which are fed in the classification process. The sorting process is applied following instructions called a *classification schedule* and uses the yard's track facilities to arrange the cars in different compositions and orders. Connected to engines, these compositions eventually leave the classification yard as *outbound trains*.

Call for Improvement

The typical freight car is reported to spend an average of 62 % of its lifetime in classification yards and only 6.6 % in service [Kum04], which calls for fundamental optimization of the classification process. In their recent article [HL10], Heydenreich and Lahrmann see significant opportunities to improve the performance of wagonload traffic. As the key points to improve efficiency and profitability, they particularly name a higher utilization of freight cars and shunting engines as well as an increased efficiency of classification yards. Also, in an interview with the Executive Vice President and Chief Operating Officer of *Canadian Railways* [Har00], reducing the dwell time of cars in railway yards to increase their utilization is identified as one of the key factors to improve railway service.

These dwell times may be reduced by accelerating the classification process, which can be achieved by minimizing the amount of movement of cars and shunting engines. It would be too expensive to extend or redesign classification yards that were designed decades ago to accommodate traffic requirements substantially different from today. An obvious way to increase the performance of existing yards is to improve the train classification process itself. The methods applied today are mostly designed from experience and may yield practically acceptable solutions. However, they offer a lot of potential for optimization. To this aim, the underlying combinatorial structure of the practical problem must be understood in order to provide efficient algorithmic solutions, to prove their solution quality, and to show that these solutions work in practice.

In this thesis a suitable theoretical model for the train classification problem is provided, which is applied to develop algorithmic solutions. The rich variety of variants of abstract problems, resulting from numerous constraints occurring in the multifaceted practical problem, as well as the arising robustness questions yield an interesting field to investigate novel algorithmic methods. Furthermore, reversely applying the gained insights to real-world railways provides a fundamental step towards a train classification process significantly more efficient than today's methods.

Summarizing, train classification presents both an interesting theoretical field of computer science research and, as it often is the bottleneck in freight transportation, an important practical problem with a high potential for improving railway freight transport.

Thesis Outline and Summary of Results

The reader is introduced to train classification in Chapter 1. First, the practical railway problem is illustrated in detail in Section 1.1. This includes the common arrangement of the track installations in classification yards in Section 1.1.1, including possible variations of the track layout. Section 1.1.2 describes the way in which classification yards are basically operated, which retrospectively motivates but also results from its design. The current classification practice is then presented in Section 1.1.3 with the most important sorting methods applied

today. In particular, different sorting requirements of outbound traffic are highlighted since they give rise to applying different classification methods in the first place. The main objective of a train classification process is given by the time it takes to complete it. This is illustrated in Section 1.1.4 among further secondary objectives. The section also introduces the reader to the various constraints occurring in practice. On the one hand, there are operational constraints, such as the departure times of outbound trains that must be met, and there is the infrastructure on the other. In particular, the track space is clearly limited in every classification yard, which concerns the lengths and numbers of available tracks.

The practical problem is then transfered to a theoretical model in Section 1.2. A novel representation of classification schedules, which present solutions of the train classification problem, is given in Section 1.2.3. In this representation each handling instruction of a car during a classification process is encoded by a bitstring, which uniquely determines the course of a car through the yard facilities. The encoding is applied in Section 1.2.4 to characterize feasible schedules. For a schedule to be feasible, the assigned bitstrings have to satisfy particular conditions for every pair of cars, but it suffices to impose this relation only for a certain subset of pairs of cars. The established necessary and sufficient feasibility conditions are then used to derive feasible schedules. In Section 1.2.5 this finally leads to a polynomial-time algorithm for computing feasible classification schedules of minimum length. This algorithm only considers the order requirements of the outbound trains, which presents the most fundamental version of the train classification problem.

Chapter 2 takes a closer look at the lengths of the involved tracks as one of the most important constraints. Considering this as an individual constraint was proved to result in an $\mathcal{NP}$-hard problem in [JMMN11]. In order to still derive optimal classification schedules, an integer programming model is established in Section 2.3. Based on the schedule representation by bitstring assignments, this model covers the order requirements for feasible schedules developed in Section 1.2.4 and contains constraints that handle the setting with tracks of limited lengths. Then, a polynomial-time 2-approximation is derived in Section 2.4, including two heuristically improved variants mak-

ing use of there being multiple outbound trains. All the versions of this algorithm inhere a condition that allows, though not necessarily, identifying calculated approximate solution as being optimal. The four approaches, and a further very simple heuristic algorithm, are compared w.r.t. their suitability and performance in an extensive experimental study in Section 2.5. The evaluation regards different objectives, including how frequently the necessary optimality condition applies for the different approximation variants, for real-world traffic data as well as a big number of synthetically derived classification problems.

Several other restrictions, individually as well as integratedly, are studied in Chapter 3. First, as the second spacial restriction, a limited number of tracks available for a classification process is treated in Section 3.2. This restriction of the practical problem is translated in a characteristic of binary schedule encodings. In contrast to tracks of limited length, a polynomial-time algorithm is provided for this setting and proved to be optimal. Then, a setting is dealt with in Section 3.3 in which parallel sorting procedures can be applied by dividing the yard and partitioning the set of cars according to their outbound train membership. Moreover, the operational constraints of train departure times and a partially time-dependent amount of available track space are considered. These problem variants are shown to be simple to solve optimally when regarded as sole restrictions each. Section 3.4 shows how they can all be incorporated into the integer programming approach mentioned above. The resulting model is tested in Section 3.5 on traffic data of a real-world classification yard, for which a schedule is computed that actually improves on the one that was originally applied. The improved schedule is finally analyzed in a thorough computer simulation in Section 3.5, which shows its applicability in practice.

Finally, in Chapter 4 the robust counterpart of the basic train classification problem is studied. The precise model is given in Section 4.1, which uses the general notion of *recoverable robustness*, that allows responding to disruptions with certain means of recovery to turn a solution back feasible. The recovery action specific to the problem model consists in including a restricted number of additional sorting steps, and a schedule is called *recovery-robust* if, for every allowed disruption, a fea-

sible schedule can be obtained from it by applying the recovery action. This is defined in Section 4.2, followed by proving an optimal strategy of where and how to apply the recovery, i.e. to insert additional steps into a recovery-robust schedule. Unfortunately, finding an optimal recovery-robust schedule presents an $\mathcal{NP}$-hard optimization problem as shown in Section 4.3. Nevertheless, for a simple but realistic setting excluding extreme disruptions, a polynomial-time algorithm for computing optimal recovery-robust solutions is developed in Section 4.4. This algorithm is evaluated experimentally in Section 4.4.4, which shows how the algorithm can be applied to vary between fast classification procedures and higher degrees of robustness. The chapter closes with a generalization of the robust problem setting to partially indistinguishable cars in Section 4.5, a setting which extends the solution space in general. For the special case of singleton trains, an efficient algorithm is derived for computing optimal recovery-robust train classification schedules.

Chapter 1

Basic Train Classification

This chapter introduces the optimization problem considered in this thesis. The practical problem setting and its variants are presented in Section 1.1, including the current operational practice and related work from the field of railway engineering. Section 1.2 first highlights the theoretical research done in this area, proceeds with the abstraction of the problem and solution techniques, and finishes with basic algorithmic results.

1.1 Practical Railway Problem

This section introduces the reader to different aspects of the train classification problem and shows how it is dealt with in practice today. The last subsection mentions some further problems closely related to train classification.

1.1.1 Classification Yard Layout

The typical layout of classification yards is shown in Figure 1.1: there is a *receiving yard*, where all inbound trains with cars to be classified arrive. Then, there is a *classification bowl*, where cars are sorted, and a *departure yard*, where sorted trains are prepared for their departure from the yard. Each such subyard mainly consists of a set of parallel railway tracks. The connec-

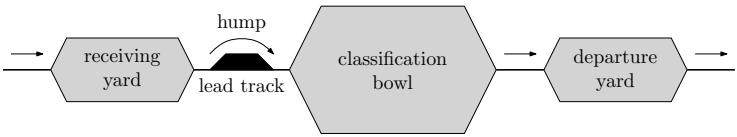
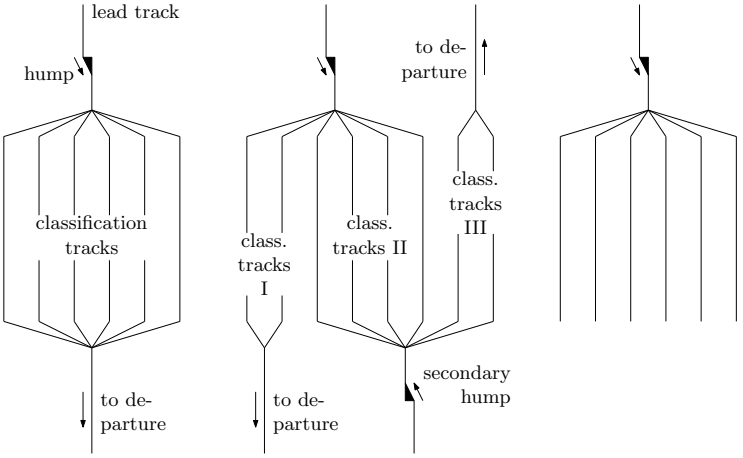


Figure 1.1: General layout of a typical classification yard with receiving subyard, classification bowl, and departure subyard.

tion between the receiving yard and the classification bowl is called *lead track*. Between the end of the lead track towards the bowl and the actual *classification tracks* in the classification bowl, there is a tree of switches by which every classification track can be reached from the lead track and vice versa. The classification tracks are connected at their other end via another tree of switches to the departure yard, where trains are brought before they leave the yard.

There are also variations of this structure, usually caused by a lack of sufficient room for the track installations. The Swiss classification yard Lausanne Triage, for example, which is regarded in more detail in Chapter 3 features no departure yard, and finished trains depart directly from the classification bowl. As another example, in the port rail of Stuttgart, Germany, the receiving yard also serves as the departure yard.

Some yards feature a hump (a rise in the ground) at the end of the lead track towards the classification bowl, in which case the yard is called a *hump yard*. A train can be taken to the lead track, where all its cars are decoupled and slowly pushed by a shunting engine over the hump. The cars then accelerate by gravity and roll into the classification bowl without any further use of an engine. Hump yards contrast to *flat yards*, which have no hump, what requires cars to be hauled by shunting engines to sort them. The more efficient hump yards are gradually replacing the less efficient flat yards [DDH83]. The considerations in this thesis generally refer to hump yards. The typical classification bowl layout of a hump yard is shown in Figure 1.2(a), with access to the classification tracks from both their ends as described above. There are also more advanced layouts, such as the one shown in Figure 1.2(b) with a secondary hump at the opposite end of the bowl. In any case, almost every yard has the layout shown in Figure 1.2(c) as a substructure, which



(a) double-ended classification bowl

(b) advanced yard layout with secondary hump

(c) core structure of hump yard

Figure 1.2: Different layouts of classification bowls in hump yards.

is essential for the sorting processes described here. Overviews of classification yard layouts can be found in [Pet77a] and their technical implementation in [Kum04].

1.1.2 Basic Yard Operation

For every car the sorting procedure in a hump yard starts by its inbound train being taken from a track in the receiving yard to the lead track, where all its cars are decoupled and pushed over the hump. At the moment a car has completely traversed the hump, the preceding car has gained some distance from the hump by rolling towards the bowl, causing a little gap between the two cars. This allows resetting the tree of switches between every pair of consecutive cars, so every car can be guided to a different classification track individually. By this process, called a *roll-in operation* or simply *roll-in*, the cars on the lead track can be separated from each other to different classification tracks.

The cars on a classification track can be pulled back over the hump on the lead track in order to perform another roll-in operation with them. This movement will be called a *pull-out (operation)* or just *pull (operation)*.

As mentioned before, the classification tracks may be connected at their opposite end via another tree of switches. This may be used to haul cars with a shunting engine from a classification track towards the departure yard. In particular, this happens for a finished outbound train on a classification track. However, an outbound train may also be *assembled* from cars on several classification tracks. Unless otherwise mentioned a double-ended classification bowl is assumed as the standard structure. For this layout assembling outbound trains at the one end of the bowl can be performed independently of the pull-out and roll-in operations at the other.

1.1.3 Sorting Requirements and State-of-the-Art Methods

Every inbound train arriving at a classification yard usually carries cars assigned for many destinations, whereas the outbound trains to be formed consist of cars for only a few destinations. A set of cars that share the same destination in an outbound train are called a *group*. All the cars of a group occur consecutively in the outbound train, whereas their relative order within the group is arbitrary. The composition of an outbound train is defined by its groups and, optionally, the order of its groups. There are different order requirements for freight trains in practice, and the respective requirement affects the choice of the sorting procedure in the hump yard of origin of a train. A systematic framework for order requirements of outbound trains is given by Dahlhaus et al. in [DMMR00]. These requirements are summarized by Hansmann et al. in [HZ07], which also contains a wide framework of sorting procedures. Further overviews of train classification can be found in [DSMM$^+$07] and [GMMW09].

The simplest case of a sorting requirement is presented by a *unit train*, the cars of which all share the same destination, such as a neighboring classification yard, and may have an arbitrary order. Such a train is simply built by reserving a classification track for it and successively rolling in the cars for this train to that track. When all inbound trains containing cars for this unit train have been rolled in, it can leave the classification yard for its destination. A unit train may be too long to be formed on a single classification track, in which case its cars are rolled in to

two or even more classification tracks. As described at the end of the last paragraph of the previous section, the cars will then be collected from the different classification tracks at their end opposite of the hump and assembled into one outbound train on a track of the departure yard.

This technique of assembling the cars of different tracks into one train may also be used for making outbound trains with multiple destinations, i.e. with multiple groups. This classification method is called *single-stage sorting* [DDH83, DHMR00, DSMM+07]. For its most basic form, there is one classification track reserved for each group of the outbound train, and the cars are rolled in to the tracks of their respective groups. When all groups are complete, the groups are assembled—in the required order if there is any—into an outbound train. Forming unit trains can be regarded as a special case of single-stage sorting with only one group per train. Unit trains are usually used for large-volume traffic such as traffic between classification yards.

For trains with many groups, such as a local freight train serving several stations along a line, single-stage sorting requires too many tracks. Also, if there are several smaller groups or even some as small as one car, a lot of idle space is generated by this method. In order to overcome this, *multistage sorting* methods can be applied, which require more car movement but make more efficient use of the available track space [Kre62, Sid72, DDH83]. Basically, the cars are pushed over the hump more than once: after the initial process of collecting cars on the classification track through consecutive roll-ins, which is also called *primary sorting*, has been finished, a shunting engine pulls the cars on a classification track back over the hump to the lead track. They can then be rolled back in, again guiding each car individually to a different classification track. This alternating process of pulling out and rolling in is also called *secondary sorting* and is iterated until all outbound trains are completed. By the term *classification schedule*, a description of any multistage sorting process will be referred to, which always includes the number and order of classification tracks pulled as well as the route of every car through the classification yard. This thesis focuses on the classification method of multistage sorting, from which a wide range of interesting optimization problems arise.

The classification methods of multistage and single-stage sorting, particularly formation of unit trains, are usually performed in a concurring manner on the same infrastructure. Over a period of 24 hours, inbound trains constantly arrive at the classification yard with cars for different outbound trains. These cars are continually rolled in to the tracks of unit trains as well as for primary sorting as part of a multistage classification process. This is stopped at some point, usually once or twice a day, after which the hump is exclusively used for secondary sorting for a few hours. Still, outbound trains not involved in secondary sorting repeatedly leave the classification bowl towards the departure yard, which also holds for outbound trains finished during the multistage sorting process. The corresponding emptied tracks may well be included again in the ongoing multistage sorting process, particularly for forming outbound trains as suggested in [Kre62]. Upon completion of the secondary sorting stage, the classification yard's mode of operation may return to primary sorting. If there is a second hump installed as depicted in Figure 1.2(b), it may be used for secondary sorting, so the primary hump will not be blocked. In this way, cars for unit trains can still be rolled in to some tracks while secondary sorting is in progress on others. Such a secondary hump is featured, for instance, by the Swiss classification yard Zürich-Limmattal, for the construction of which Baumann [Bau59] explains the design aspects concerning multistage sorting. The secondary hump in the resulting layout, however, is currently not used due to cost and organizational reasons [Hol07]. Other variations from the standard hump yard layout may result in minor deviations of the local operating characteristics, which does not change the essential yard operation principle.

There are different multistage methods commonly applied today, varying in the strategy they roll in cars and apply secondary sorting. The two cardinal multistage methods *sorting by train* and *simultaneous sorting* are compared by Krell [Kre62, Kre63], including two improvements of the latter called *triangular sorting* and *geometric sorting*, respectively. Sorting by train and the simultaneous sorting are also described in different fashions by Flandorffer [Fla53], by Pentinga [Pen59], and by Siddiqee [Sid72]. Boot [Boo57] describes the operational constraints when simultaneous sorting was introduced in France,

Belgium, and The Netherlands. Some of these methods are again considered by Daganzo, Dowling, and Hall in [DDH83], who particularly analyze triangular sorting with regard to the differences when applied in hump and flat yards. The mentioned four multistage methods will be described in more detail in Section 1.2.6. This will be done by means of the schedule representation introduced in Section 1.2.3, by which a more efficient description is achieved.

What is common in all these single- and multistage classification methods is the fact that they do not consider the order in which the cars arrive in the inbound trains—with the exception of the example given in [Kec58], which is explained in Section 1.2.6. This practice may yield a sorting procedure in which the course of every car is independent of its position relative to the other inbound cars; however, this must be paid for by a potential waste of valuable resources, such as a high number of required classification tracks or pull-out operations. These resources are highlighted in the following section, among other constraints and objectives.

1.1.4 Objectives and Constraints

The most important objective of a multistage sorting process is the time it takes to complete it. This time is contributed to by the pull-out and the roll-in operations. Pulling out a track takes an amount of time c_{pull} which is determined by the distance the shunting engine must drive in this yard. The time to roll-in the cars from the lead track is proportional to their number and depends on the time c_{push} required for decoupling and pushing a single car. Together, a multistage sorting process of h pull-out steps and a total of r cars rolled in approximately requires a time of $hc_{\text{pull}} + rc_{\text{push}}$. Increasing the value of h allows reducing r and vice versa. For a typical classification yard, however, the first summand dominates the second, and the main objective throughout this thesis will be the number of pull steps h. Section 1.2.5 introduces a novel algorithm to find classification schedules that minimize h, which also shows that the traditional methods presented in Section 1.2.6 yield suboptimal schedules in general.

Still, it is also common practice to consider the total number of cars rolled over the hump as an objective, which is done, for

example, at some classification yards for pricing classification requests of customers. The total number of roll-ins r will play an important role in the algorithm of Section 2.4.1 and occurs as a secondary objective in Sections 2.3, 3.4, and 3.5. Besides, setting the switches between the lead track and the classification bowl contributes to their wear. The number of settings of switches is a minor objective to which attention is paid in Section 3.5.

As mentioned in the previous section, the track space in classification yards must be utilized carefully as it is a tightly limited resource. This particularly holds for the number of available classification tracks, a number of which is usually reserved for multistage sorting in practice, while the rest is used for other sorting activities. An early example for how to deal with a restricted number of classification tracks is given by Krell [Kre62]. The author also gives some upper bounds on the numbers of groups that can be sorted, not considering the inbound order of cars, at which a closer look is taken in Section 1.2.6. Rough requirements of the different multistage methods on the number of classification tracks are also mentioned in [Pen59, Sid72, DDH83]. A precise approach for finding multistage sorting schedules under a limited number of tracks can be found in [HZ07].

The length of the classification tracks presents another restriction, which is given by a highest number of cars a track may accommodate. This is dealt with, e.g., in [Pet77a, Pet77b], where track congestions resulting from restricted track capacities are considered for single-stage sorting. For the different multistage sorting methods, several publications note that there exist certain capacity requirements without precisely deriving them [Boo57, Pen59, Sid72]. In order to level the required number of tracks and the track occupation of the individual tracks in the primary and the secondary sorting stage, an approach of treating several outbound trains as one is introduced in [Dag86, Dag87a, Dag87b], which is called *convoy formation*.

Apart from the number and length of classification tracks constraining the main objective h, there may further exist specific infrastructural or operational constraints to the classification process at single classification yards. As an example, in the advanced layout of Figure 1.2(b), the classification track on

which an outbound train is formed in a multistage sorting process involving the secondary hump may not be accessible from the primary hump. Then, not even the car at the head of this train can be sent to that track in primary sorting. Some characteristics of the four methods mentioned in Section 1.1.3 above are summarized in [Sid72] with regard to which method is beneficial under which circumstances. Endmann [End63] as well as Krell [Kre63] consider the distribution of arrival and particularly departure times of freight trains at classification yards with regard to the suitability of different classification methods in Germany in the 1960s. Departure times of outbound trains are also considered in [BFMM11] amongst various other practical constraints for a related track allocation problem in classification yards. Departure time restrictions are also included in the approach presented in Sections 3.3–3.5.

1.1.5 Related Concepts and Problems

Closely connected to the above introduced concept of groups is the notion of a block in a network of classification yards [CTV98]: a *block* is a set of cars that take a common itinerary over several classification yards and is not broken up at intermediate yards. Both concepts are related in the following way: before a block starts its journey through the network, it must be built at some origin classification yard. The block may be built in a multistage classification process, where it can be regarded as a group of an outbound train and formed accordingly. If a block arrives at an intermediate yard, it is not broken up and can be regarded as a single car in the classification task. Finally, if a block reaches its destination yard, it will just be broken up as part of an inbound train. Hence, from the point of view of a single classification yard, the concept of blocks does not enhance the local sorting problems. From a railway network point of view, there are various related optimization problems, such as blocking [BGSR80], routing and train scheduling [HBSM95] or block-to-train assignment [JAŞ08]. These problems are out of the scope of this thesis, an overview can be found in [CTV98].

Closely related to train classification is the problem of sorting with networks of stacks. Knuth explains the mechanism of stacks, queues, and deques in [Knu97] in terms of an analogy from classifying railway cars. Inspired by this, Tarjan formu-

lates the problem of sorting with networks of stacks or queues in [Tar72]: there is a directed graph in which every node is associated with a stack or a queue. There is a special source node, where a sequence of items is located initially. These items may successively be moved along edges through the network in order to obtain a sorted sequence at the dedicated target node. Train classification can be regarded as a special variant of this problem: corresponding to the roll-in and pull-out operations, a node has to be emptied completely before any car is moved from any other node. A survey of combinatorial results for different variations of this problem is given by Bóna in [Bón02]. An algorithmic approach for minimizing the movements of cars along edges is taken by König and Lübbecke in [KL08].

1.2 Theoretical Solution Approach

A theoretical model for the practical train classification problem introduced in Section 1.1 is established in this section. The relevant theoretical research is reviewed in Section 1.2.1, followed by various notations and a definition of the precise setting in Section 1.2.2. Then, in Section 1.2.3 an efficient representation of classification schedules is presented. This is applied for the most basic setting to derive feasible schedules in Section 1.2.4 and optimal schedules in Section 1.2.7.

1.2.1 Related Work

Compared to the number of publications from railway engineering, which are presented in Section 1.1 above, only little algorithmic work related to multistage sorting has been done. Beside the mentioned surveys [DSMM$^+$07] and [GMMW09] giving a rather algorithmic perspective, the most important theoretical publications are from Dahlhaus et al. [DHMR00, DMMR00, Dah08] and Hansmann and Zimmermann [HZ07]. For the method of single-stage sorting, Dahlhaus et al. show that determining the minimum number of required tracks is an $\mathcal{NP}$-complete problem in [DHMR00]. This is obtained for a sorting requirement in which not only the relative order of cars within the group is arbitrary but also the order in which the groups appear in the outbound train. Assuming the same order requirement and

no further constraints, this result implies that minimizing the number of steps for multistage sorting is $\mathcal{NP}$-hard too, which is shown in [DHMR00].

A related algorithmic multistage sorting problem is considered in [DMMR00], in which a multistage method is applied that resembles the radix sort algorithm. Their definition of a sorting step differs from the one assumed here, as not only one but *all* classification tracks are pulled out to the lead track before the next roll-in operation is performed. They show how to optimize their procedure for several order requirements, particularly including the case of a fixed order of groups in the outbound train.

1.2.2 Abstract Setting and Notation

The definitions and assumptions made in this section are used in all chapters unless otherwise mentioned. In several subsequent sections, additional notation will be introduced when required for the respective considerations there.

Cars and Trains

Any *car* involved in the classification process is represented by a positive integer number $\tau \in \mathbb{N}$, which will also be called the *type* of the car. A *train* is represented by a k-tuple $(\tau_1, \ldots, \tau_k)$ of cars $\tau_x \in \mathbb{N}$, $x = 1, \ldots, k$, where k will also be called the *length* of the train. Car τ_1 will also be called the *first* car of the train, and τ_k the *last* car. For two cars τ_x, τ_y of a train, τ_x will be said to be *in front of* τ_y or to *occur before* τ_y if $x < y$. If a train is located on the lead track, the car closest to the hump presents the first car. If a train is located on a classification track, the car closest to the dead-end or the exit presents the first car of the train. Note that the engine is not included in the definition of a train and may be at either end of the train depending on the situation.

In every instance of a classification problem, there is a sequence of inbound trains given, the number of which is denoted by ℓ. Each such inbound train has an arrival time at the classification yard. The concatenation of the trains in the order they arrive will be called the *inbound sequence of cars*. The inbound

trains will be given in a fixed order, with the exceptions of Section 1.2.7 and Chapter 4: Section 1.2.7 considers the problem of finding an optimal inbound order of trains, while Chapter 4 deals with handling deviations from the expected inbound order. As its second ingredient, any instance of a classification problem contains m order specifications corresponding to m outbound trains. The problem task is to find a multistage sorting process that, when applied to the inbound sequence of cars, yields m outbound trains, each one on a separate classification track, all matching their given order specifications.

Types

The above-mentioned concept of type is used to represent common characteristics of cars. This will usually be a common destination. In this case the type resembles a car's group membership, which determines the car's position in its outbound train (Section 1.1.3).

Let $T_k = (\tau_{k,1}, \ldots, \tau_{k,n'_k})$ be the kth inbound train, where n'_k denotes its length. Then, let the *maximum type* G be defined by $G := \max_{1 \le k \le \ell} \max_{1 \le i \le n'_k} \tau_{k,i}$. The number of cars of a type t in the inbound sequence of cars will be denoted by n^t. In general, the cars are assumed to be unique w.r.t. their type, i.e., $n^t = 1$ for every type $t = 1, \ldots, G$. The remainder of this chapter, Chapter 3, and Section 4.5 present exceptions from this: as noted in Section 1.3 below, [JMMN07] only addresses unique cars and the considerations of [JMMN11] only touch on types of cars. The most fundamental results from there are effectively proved to hold also for cars with types in Sections 1.2.3–1.2.5. The considerations of Chapter 3 hold independently of types of cars. Section 4.5 deals with the generalization of the robustness considerations of Sections 4.1.2–4.4 to the setting with types of cars.

The following assumption for the inbound sequence of cars is always made: for every type $t \in \{1, \ldots, G\}$ there is a car $\tau_{k,i} = t$ in the inbound sequence of cars, i.e., every single type from 1 to G does occur. Besides, since there may be several cars of the same type, the cars are implicitly assumed to carry indices according to their relative inbound order if they are referred to by their type (instead of their exact position in the inbound se-

quence of cars): if $\tau_{k_1,i_1} = \tau_{k_2,i_2} = \ldots = \tau_{k_{n^t},i_{n^t}} = t$ are all the cars of some type $t \in \{1,\ldots,G\}$ in the inbound sequence and these cars arrive in just this order $\tau_{k_1,i_1}, \tau_{k_2,i_2}, \ldots, \tau_{k_{n^t},i_{n^t}}$, then t_1 will refer to car τ_{k_1,i_1}, t_2 to car τ_{k_2,i_2}, etc. This notational alternative will ease reasoning at several points such as in Lemma 2 or Theorem 2 further below in this chapter. Usually, the type-independent notation $\tau_{k,i}$ explicitly referring to the exact position of a car in the inbound sequence of cars is used though. From now, this will always happen in a simplified form by considering the concatenation $\tau_1,\ldots,\tau_n$ of all inbound trains and referring to the xth car of the resulting inbound sequence of cars by τ_x.

For an example consider the inbound sequence of $n = 8$ cars $4,3,4,2,3,1,3,3$. The maximum type is $G = 4$, and the numbers of cars of the different types are $n^1 = 1$, $n^2 = 1$, $n^3 = 4$, and $n^4 = 2$. The first arriving car has type 4, i.e. $\tau_1 = 4$, the second arriving car has type 3, i.e. $\tau_2 = 3$, the third arriving car has type 4 again, i.e. $\tau_3 = 4$, etc. In the alternative notation, 4_1 refers to car $\tau_1 = 4$ as it is the first arriving car of type 4, and 4_2 refers to car $\tau_3 = 4$ as it is the second arriving car of type 4. 3_1 refers to $\tau_2 = 3$ in this notation as it is the first car of type 3 to arrive, 3_2 refers to $\tau_5 = 3$, 3_3 to $\tau_7 = 3$, and 3_4 to $\tau_8 = 3$. Finally, 2_1 refers to $\tau_4 = 2$, which is the only car of type 2, and 1_1 refers to $\tau_6 = 1$.

The order specifications of the outbound trains are always assumed to have the following form: cars of the same type belong to the same outbound train, and all the types of a train are consecutive. More precisely, for any pair of cars τ_x, τ_y with types $\tau_x = t \leq t' = \tau_y$, if τ_x and τ_y belong to the same outbound train, then every car τ_z with $t \leq \tau_z \leq t'$ belongs to this outbound train as well. In terms of the notion of groups of Section 1.1.3, this means that no destination occurs in two outbound trains and the groups of each train have a fixed order. Moreover, w.l.o.g. the order of cars in an outbound train is nondecreasing w.r.t. the cars' types, i.e., if a car τ_x occurs before $\tau_{x'}$ in the same outbound train, then $\tau_x \leq \tau_{x'}$. These assumptions about the order requirements come under the case of "Q-node with P-children" in the terminology of [DMMR00], "ordered g-blocks" in the terms of [HZ07], and a "single output string" (which is a "twinning string") in [DSMM$^+$07].

Classification Tracks

There is one classification track reserved for the final formation of each outbound train. Every such track is also called an *output track*. All other classification tracks are used for the actual sorting procedure and are also called the *sorting tracks*. Each classification track either serves for sorting or output but not both, and the number of output tracks is assumed to meet the number of outbound trains. As an exception, Sections 3.3–3.5 incorporate an initially restricted number of available output tracks that increases during the classification process, which includes pulled sorting tracks to be reused as output tracks. W.l.o.g. every involved sorting track is pulled exactly once. If there is a limited number W of sorting tracks, it may happen that the sorting cannot be accomplished by filling and pulling each sorting track at most once. Then, some sorting tracks must be filled and pulled several times during the classification process. This special situation is dealt with in Section 3.2.

The highest number of cars a track may accommodate will be called the *capacity* of the track. The classification tracks have uniform capacities, and the capacity of a single track is denoted by C. Capacities are assumed to be unrestricted for the remainder of this section and in Chapter 4. Chapter 2 treats restricted track capacities in detail, which is also incorporated in Sections 3.4 and 3.5 of Chapter 3.

Schedules

In order to specify a single pull-out operation, it suffices to specify the sorting track that is pulled since all the cars on it are pulled. To specify a roll-in operation, a target classification track must be given for every car on the lead track. A pair of a pull-out and a subsequent roll-in operation is called a *sorting step* or just *step*. An initial roll-in operation followed by a sequence of sorting steps is called a *classification schedule* and the number of steps is called its *length*. For a schedule of length h, the single sorting steps will usually be indexed from 0 to $h-1$ and referred to by the ith step, $i = 0, \ldots, h-1$. A classification schedule B is called *feasible* if its application to the inbound sequence of cars yields the correctly sorted outbound trains, each on an arbitrary but separate output track.

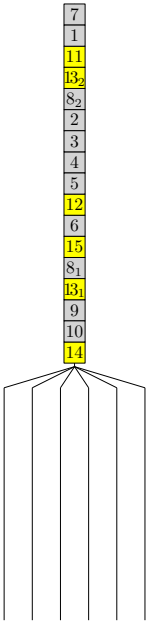
Objectives

As mentioned in Section 1.1.4, the number of sorting steps presents the main objective in general, so the task is to find a feasible classification schedule of minimum length. The total number of cars rolled in plays a role as a secondary objective for several considerations later in this document, such as for the integer programming approach of Section 2.3 or 3.4. For a schedule B of length h, the *weight of* B is defined as the total number of roll-ins of cars, excluding the final roll-in of every car, when executing B. It will be denoted $w(B)$.

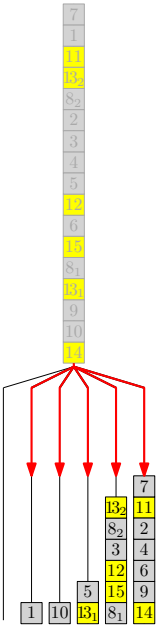
1.2.3 Classification Schedule Representation

For any classification process of h sorting steps, the course of any car over the classification tracks can be represented by a binary string $b = b_{h-1} \dots b_0$ with $b_i = 1$ if and only if the car visits the sorting track pulled in the ith sorting step, $i = 0, \dots, h-1$. In the example of Figure 1.3, car 7 visits the track pulled in the first and third of four steps, so the corresponding bitstring is given by $b = 0101$. In this way, every classification schedule of length h can be represented by an assignment of cars to bitstrings of length h. For the example this is done in Figure 1.3(l). The order in which the bitstrings are listed is explained in the Section 1.2.4. Note that there is no need to explicitly include the roll in of a car to an output track in the encoding because every car will finally be sent to its corresponding outbound train independently of the form of the preceding classification process.

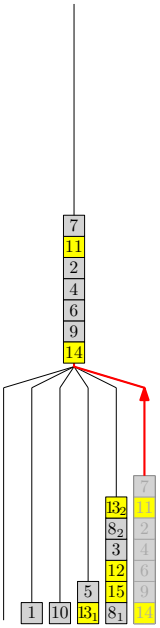
Classification schedules will henceforth be represented by such a bitstring assignment for every car. In a given schedule B, the bitstring to which a car τ_x is assigned will be referred to by b^{τ_x}. In order to execute a given schedule B, a car is initially rolled in simply to the sorting track that is the first one to be pulled among all tracks that the car visits during the classification process. In other words, car τ_x is sent to the kth sorting track for $k = \min_j \{j < h \mid b_j^{\tau_x} = 1\}$. In the example of Figure 1.3, car 7 is sent to the track pulled first, whereas car 10 with $b^{10} = 1000$ goes to the fourth track. If $b_j^{\tau_x} = 0$ for all $j = 0, \dots, h-1$, then τ_x is directly sent to the hth track, i.e. the output track of the outbound train to which it belongs. This happens for car 1 as $b^1 = 0000$ for the schedule of Figure 1.3.



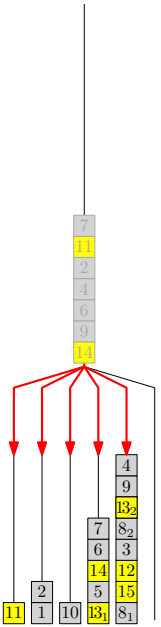
(a) inbound cars on lead track



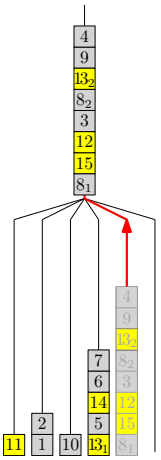
(b) initial roll-in



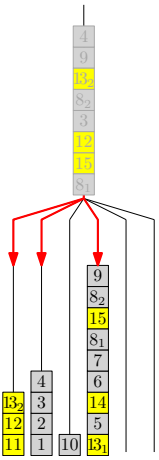
(c) 1st pull-out initiating the 1st sorting step



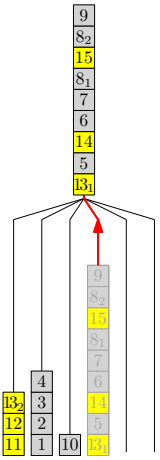
(d) 2nd roll-in completing the 1st step



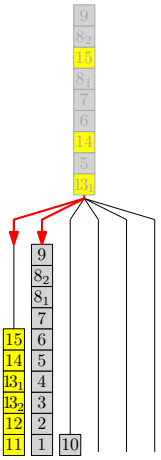
(e) 2nd step with pull-out...



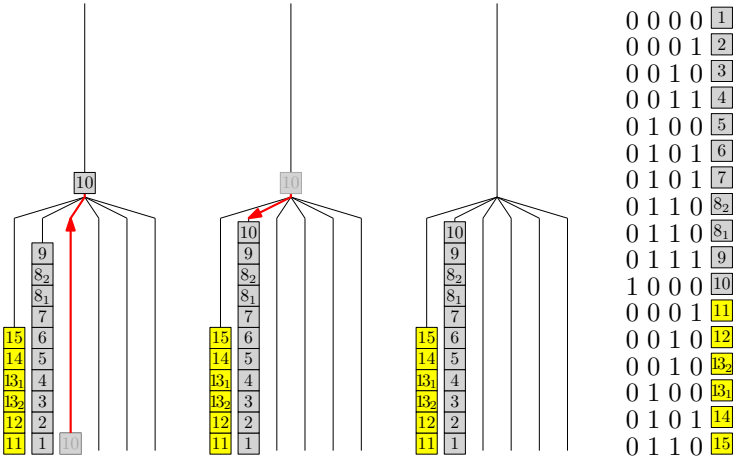
(f) ...and roll-in



(g) 3rd pull-out



(h) 4th roll-in finishes an outbound train



(i) last pull-out operation

(j) final roll-in completing the schedule

(k) outbound trains made on separate tracks

(l) bit-string encoding of schedule

Figure 1.3: An example of a classification process with an inbound sequence of $n = 17$ cars with types, sorted into $m = 2$ outbound trains, using $h = 4$ sorting steps. Cars of the same color are members of the same outbound train. Cars with the same destination number (type) are indexed in the order they arrive as explained in Section 1.2.2 for unique references.

(This would have been a sensible assignment for car 11, i.e. the first car of the yellow train, too.) If $b_i^{\tau_x} = 1$, $i \in \{0, \ldots, h-1\}$, then car τ_x is pulled out in the ith sorting step from the corresponding ith track. After this pull-out it is rolled in to the kth sorting track for $k = \min_j\{i < j < h \,|\, b_j^{\tau_x} = 1\}$, i.e. to that sorting track the car is indicated to visit next. For instance, after car 7 has been pulled in the first step because $b_0^7 = 1$, it is sent to the track pulled third as $b_1^7 = 0$ but $b_2^7 = 1$. Also here, if there is no such bit $b_j^{\tau_x} = 1$ with $i < j < h$, the car is sent to its output track.

For a classification schedule B of length h that is given in this bitstring representation, different characteristics of the procedure can be read off directly from the bitstring assignment. As a simple example, if the all-zero bitstring is assigned to any

car in a schedule B, i.e., $b^{\tau_x} = 0 \dots 0$ for some car τ_x, this car is initially sent right to the output track of its outbound train. Such a schedule can only be applied if the respective output track can be directly accessed from the primary hump as mentioned at the end of Section 1.1.4. Moreover, the weight of the schedule B can be obtained by simply counting the number of bits b_i^j of B with $b_i^j = 1$:

$$w(B) = \sum_{i=0}^{h-1} \sum_{j=1}^{n} b_i^j$$

This corresponds to the total number of roll-ins except for the final roll-ins of cars to their respective output tracks as mentioned above. The number of roll-ins in the ith sorting step corresponds to the number of cars pulled out in this step, $i \in \{0, \dots, h-1\}$. If the schedule B is regarded as a bit table with the bitstrings $b^1, \dots, b^n$ written one below the other, as done in Figure 1.3(l) for the example, this number is determined by counting the number of bits in the ith column b_i with $b_i^{\tau_x} = 1$ and will be called the *weight of column* b_i. It is denoted by $w(b_i)$ and calculated as $w(b_i) = \sum_{x=1}^{n} b_i^{\tau_x}$.

In addition to those mentioned here, some more characteristics of schedules hidden in the representation will be highlighted in Chapters 2 and 3. They will actually not be introduced as characteristics of a given schedule but as restrictions on a schedule that is to be derived for a given train classification problem instance. A schedule must then satisfy these restrictions in addition to the main feasibility conditions developed in the following section.

1.2.4 Feasible Schedules

An assignment from cars to bitstrings has been introduced in the previous section to encode classification schedules. This assignment can conversely be applied to derive feasible schedules, which is explained in this section. Note that "feasible" only refers to the correct order of outbound trains here as defined in Section 1.2.2, additional conditions will only apply in the subsequent chapters.

Every nonnegative integer k has a unique binary representation $[k]_2$ of $\lceil \log_2 k \rceil$ bits, just as every bitstring $b = b_{h-1} \dots b_0$

of length h represents a unique nonnegative integer $\sum_{i=0}^{h-1} 2^i b_i$. This yields an order relation for bitstrings:

$$b < b' \quad :\Leftrightarrow \quad \sum_{i=0}^{h-1} 2^i b_i < \sum_{i=0}^{h-1} 2^i b'_i \tag{1.1}$$

If b and b' differ in length, then leading zeros are implicitly added at the left end of the shorter bitstring, so the order relation is well-defined. If bitstrings are compared, they will always be assumed to have the same length. Further note that it suffices to compare the most significant among the bits for which the two binary numbers, i.e. the bitstrings, differ. The following lemma relates the courses of two cars in a classification process to each other according to their assigned bitstrings.

Lemma 1 *Let B be a classification schedule of length h that is applied to an inbound sequence $\tau_1, \ldots, \tau_n$ of n cars. If a pair of cars τ_x, τ_y, $x, y \in \{1, \ldots, n\}$ with $x < y$ belongs to a common outbound train, the cars swap their relative order in the outbound train if and only if $b^{\tau_x} > b^{\tau_y}$.*

Proof If $b^{\tau_x} > b^{\tau_y}$, there is some index $i \in \{0, \ldots, h-1\}$ such that $b^{\tau_x}_{h-1} \ldots b^{\tau_x}_{i+1} = b^{\tau_y}_{h-1} \ldots b^{\tau_y}_{i+1}$, $b^{\tau_x}_i = 1$, and $b^{\tau_y}_i = 0$. Hence, τ_x is pulled in the ith sorting step and sent to the kth track for $k = \min_j\{i < j < h \,|\, b^{\tau_x}_j = 1\}$. Since τ_y is not pulled in the ith step and $b^{\tau_y}_{h-1} \ldots b^{\tau_y}_{i+1} = b^{\tau_x}_{h-1} \ldots b^{\tau_x}_{i+1}$, τ_y is standing on the kth track when τ_x is sent there, so τ_y appears on that track before τ_x. Since they do not change their relative order in any later sorting step, τ_y appears on the output track before τ_x, so they have swapped their relative order compared to the inbound order.

If $b^{\tau_x} < b^{\tau_y}$, the same line of reasoning applies with reversed roles for τ_x and τ_y, so τ_x appears on the output track before τ_y, which means no change of their relative order.

If $b^{\tau_x} = b^{\tau_y}$, both cars will take exactly the same course over the classification tracks and appear on every track they visit just in the same relative order they originally arrived on the lead track. □

Consider this lemma for the example given in Figure 1.3 with $x := 10$ and $y := 12$, which yields $\tau_x = 4$ and $\tau_y = 2$ with $b^4 = 0011 > b^2 = 0001$. The most significant index for which

the bitstrings differ is $i = 1$ with $b_1^4 = 1$ and $b_1^2 = 0$, which corresponds to the pull-out step (Figure 1.3(e)) after which the two cars obtain their final relative order. As shown in Figure 1.3(f), car 4 has just been pulled and is sent to track where car 2 is standing, so the relative order is swapped. On the other hand, cars 6 and 7 do not swap because 6 arrives before 7 and $b^6 = b^7$, nor do 6 and 8_2 because 6 arrives before 8_2 and even $b^6 < b^{8_2}$.

Using this insight the following theorem contains the necessary and sufficient order requirements for a bitstring assignment to present a feasible classification schedule. Recall the assumptions of Section 1.2.2 on the order specifications of the outbound trains: cars of the same type belong to the same outbound train, and the types of a train are consecutive and nondecreasing. The feasibility conditions stated in the theorem below depend on the inbound order of the cars and their types, which will later be used to exploit presortedness in the input, i.e. an order of inbound cars where long ordered subsequences of cars occur by incident.

Theorem 1 *Let $\mathcal{T} = \tau_1, \ldots, \tau_n$ be the inbound sequence of cars of a classification instance with m outbound trains, and let B be a classification schedule of length h. B presents a feasible classification schedule for $\mathcal{T}$ if and only if both the following conditions hold for every pair of cars τ_x, τ_y of $\mathcal{T}$ that belong to the same outbound train:*

$$\tau_x > \tau_y \quad \Rightarrow \quad b^{\tau_x} \geq b^{\tau_y} \tag{1.2}$$

$$x < y \text{ and } \tau_x > \tau_y \quad \Rightarrow \quad b^{\tau_x} > b^{\tau_y} \tag{1.3}$$

Proof ($\Leftarrow$) Consider any pair τ_x, τ_y of cars of the same outbound train with $\tau_x > \tau_y$. If $x > y$, then $b^{\tau_x} \geq b^{\tau_y}$ by (1.2), so the two cars do not swap their relative order by Lemma 1 and appear in correct order on their output track. If $x < y$, then $b^{\tau_x} > b^{\tau_y}$ by (1.3), so they do swap by Lemma 1, and τ_y arrives on the output track earlier than τ_x, just as required. Hence, in every outbound train every pair of cars with different types $\tau_x > \tau_y$ will appear in correct relative order, so all outbound trains will correspond to their respective order specifications.

($\Rightarrow$) Let $\tau_x > \tau_y$ for any pair of cars $x, y \in \{1, \ldots, n\}$, $x \neq y$, of a common outbound train. If (1.2) does not hold, i.e., $b^{\tau_x} < b^{\tau_y}$, then τ_y will appear in the outbound train before

τ_y by Lemma 1. This order violates the order specification of the outbound train, so B is not feasible.

If (1.3) does not hold, i.e., $x < y$ but $b^{\tau_x} \leq b^{\tau_y}$, then $b^{\tau_x} = b^{\tau_y}$ must hold for feasibility as just proved. Hence, τ_x and τ_y take exactly the same course over the classification tracks without changing their relative order, so they will end up on their common output track in the same order they arrived in the inbound sequence of cars. Because $x < y$, τ_x ends up on the output track before τ_y, which is an order not corresponding to a feasible schedule. □

This theorem basically determines how to assign bitstrings to pairs of cars τ_x, $\tau_y = \tau_x + 1$ of consecutive types of the same outbound train: first, $\tau_x + 1$ must never get a bitstring that is strictly smaller than that of τ_x by condition (1.2). This holds, for example, for the pair of cars 11, 12 as well as for the pair 12, 13_2 of the yellow train in Figure 1.3. Second, if $\tau_x + 1$ arrives before τ_x, then $\tau_x + 1$ must even get a strictly greater bitstring than τ_x by condition (1.3). This again holds for 11 and 12, which arrive in wrong order. Cars 12 and 13_2, however, can be assigned the same bitstring as they arrive in the right order. This exactly is a point where presortedness is exploited to reduce the number of sorting steps. Moreover, there is no order relation between pairs τ_x, τ_y of the same type $\tau_x = \tau_y$ or between cars of different outbound trains.

These considerations will finally lead to a greedy algorithm for finding a feasible schedule of minimum length, which is explained in Section 1.2.5. First, a structural insight about classification schedules is given in Lemma 2. Recall from Section 1.2.2 that cars of the same type can be uniquely referred to by assuming implicit indices according to the order in which they arrive: for any type $t \in \{1, \ldots, G\}$, t_1 will refer to the first car of type t to arrive, t_2 to the second, etc. Consider the total order for sets of cars with multiple cars per type defined as follows, where t_i and t'_j are cars with $t, t' \in \{1, \ldots, G\}$ and $i, j \geq 1$:

$$t_i < t'_j \quad :\Leftrightarrow \quad t < t' \text{ or } (t = t' \text{ and } i > j) \tag{1.4}$$

This order may appear counterintuitive at first sight, its relevance is made clear in Lemma 2 further below. It corresponds to the following ordering of cars from an inbound sequence of cars: $1_{n_1}, \ldots, 1_1, 2_{n_2}, \ldots, 2_1, \ \ldots \ , G_{n_G}, \ldots, G_1$. It is also used

for the example in Figure 1.3(l). In fact, for any schedule that is given in the representation introduced in Section 1.2.3, the sequence of bitstrings corresponding to order (1.4) can be assumed nondecreasing. This is shown in the following lemma, which will be used in the proof of Theorem 2.

Lemma 2 *Let an instance of a classification problem (with potentially multiple cars per type) be given. For any feasible schedule B, there exists a feasible schedule B' of the same length in which $t_i > t'_j$ (w.r.t. (1.4)) implies $b^{t_i} \geq b^{t'_j}$ for every pair of cars t_i, t'_j of the same outbound train.*

Proof Let t_i and t'_j be cars of the same outbound train. If $t > t'$, then $b^{t_i} \geq b^{t'_j}$ by (1.2) since B is feasible. For the remainder, assume $t = t'$, w.l.o.g. $i < j$, and $b^{t_i} < b^{t'_j}$ in B. Let B' denote the schedule that is identical to B except for $b'^{t_j} := b^{t_i}$.

Let s_k be a car of type $s > t$. Then, $b^{s_k} \geq b^{t_j}$ by (1.2), and $b^{s_k} > b'^{t_j}$ since $b^{t_j} > b'^{t_j}$. Hence, feasibility is preserved by the reassignment of t_j in this case.

Let now s_k be a car of type $s < t$. Then, $b'^{t_j} = b^{t_i} \geq b^{s_k}$ by (1.3), so feasibility of B' is preserved by the reassignment of t_j if s_k arrives before t_i. If s_k arrives after t_i, then B even satisfies $b^{s_k} < b^{t_i}$ by (1.3), so inequality $b'^{s_k} < b'^{t_j}$ still holds in B'.

Cars s_k of type $s = t$ do not underly any feasibility constraints with t_j, so B' is a feasible schedule. For every pair of cars t_i, t_j with $i < j$ that belong to a common outbound train, a corresponding reassignment for t_j to b^{t_i} can be performed. Since the bitstrings of cars are only decreased in this way, the process eventually stops with a feasible schedule B' in which $t_i > t'_j$ implies $b^{t_i} \geq b^{t'_j}$ for every pair of cars t_i, t'_j of the same outbound train. □

Also the optimal schedule computed by the algorithm in Section 1.2.5 will have this structure. Essentially, the algorithm will greedily grow subsequences of the inbound sequence of cars that present presorted sequences of cars occurring consecutively in an outbound train. These subsequences are defined as follows.

Definition 1 *Let $\mathcal{T} = \tau_1, \ldots, \tau_n$, $\tau_x \in \{1, \ldots, G\}$, $x = 1, \ldots, n$, be the inbound sequence of cars for a given classification problem instance. For any pair of cars τ_x and τ_y, $x, y \in \{1, \ldots, n\}$, with*

$\tau_x < \tau_y$, let $[\tau_x, \tau_y]$ denote the subsequence of $\mathcal{T}$ that contains exactly those cars τ_z, $z \in \{1, \ldots, n\}$, that are contained in the same outbound train as τ_x and match one of the following three conditions: $\tau_z = \tau_x$ and $z \leq x$, $\tau_x < \tau_z < \tau_y$, or $\tau_z = \tau_y$ and $z > y$. Such a subsequence $[\tau_x, \tau_y]$ of $\mathcal{T}$ will be called a chain *of $\mathcal{T}$ if it is nondecreasing (i.e. the types of the cars are nondecreasing).*

Roughly speaking, a chain covers all cars of several consecutive types, but some cars of the types of its smallest and greatest car may be excepted. Note that in the definition above, τ_x is a member of the chain and presents the latest arriving car of its type that is a member. If some car τ_z with the same type $\tau_z = \tau_x$ arrives later, i.e. $z > x$, then it is not in the chain. However, τ_y is not a member of the chain, and it is just the latest arriving car of its type that is not a member. Any car τ_z of the same type $\tau_z = \tau_y$ that arrives later, i.e. $z > y$, is a member of the chain. Consider the inbound sequence of cars in Figure 1.3(a). (The example shows only little presortedness, but should suffice for illustration here.) The sequence defined by $[6, 8_2]$ contains cars 6 and 7 and presents a chain covering all cars of its smallest ($t = 6$) and greatest ($t = 7$) contained type. As another example, sequence $[12, 13_1] = (12, 13_2)$ presents a chain containing only car 13_2 but not 13_1 of its greatest type. The sequence $[6, 8_1] = (6, 8_2, 7)$, however, is not a chain as it is not nondecreasing. Note that by Definition 1 a chain never spans cars of more than one outbound train. This leads to a total assignment of cars to chains that determines the degree of presortedness of the inbound sequence of cars. This is formalized in the following definition. Recall that, by the assumptions for outbound trains in Section 1.2.2, all cars of the $(j+1)$th outbound train have a strictly greater type than those of the jth train.

Definition 2 *Let $\mathcal{T} = \tau_1, \ldots, \tau_n$, $\tau_x \in \{1, \ldots, G\}$, $x = 1, \ldots, n$, be the inbound sequence of cars of a train classification problem with m outbound trains, and let $\tau_{n+1} := G+1$. Let further $\tau_{y_1^1}, \ldots, \tau_{y_{k_1}^1}, \tau_{y_1^2}, \ldots, \tau_{y_{k_2}^2}, \ \ldots \ , \tau_{y_1^m}, \ldots, \tau_{y_{k_m}^m}, \tau_{y_1^{m+1}}$ be a sequence of cars of $\mathcal{T}$ that satisfies the following two properties for every index $j = 1, \ldots, m$: car $\tau_{y_i^j}$ belongs to the jth outbound train and $\tau_{y_i^j} < \tau_{y_{i+1}^j}$ for every $i = 1, \ldots, k_j$.*

Then, the corresponding sequence of indices (of the inbound cars) $y_1^1 \,..\, y_{k_1}^1, y_1^2 \,..\, y_{k_2}^2, \ldots, y_1^m \,..\, y_{k_m}^m, y_1^{m+1}$ *will be called a* chain decomposition *of* $\mathcal{T}$ *if it satisfies the following four conditions:*

- $y_1^1 = \max\{1 \le x \le n \,|\, \tau_x = 1\}$,
- $[\tau_{y_i^j}, \tau_{y_{i+1}^j}]$ *presents a chain for every* $j = 1, \ldots, m$ *and* $i = 1, \ldots, k_j - 1$,
- $[\tau_{y_{k_j}^j}, \tau_{y_1^{j+1}}]$ *presents a chain for every* $j = 1, \ldots, m$, *and*
- $y_1^{m+1} = n + 1$.

The length *of the chain decomposition is defined by* $\max_{j=1\ldots m}\{k_j\}$.

Note that this definition covers every car of $\mathcal{T}$, i.e., every car is contained in some chain of the decomposition. Consider the inbound sequence of cars of Figure 1.3(a) again. The notation of Definition 2 gives $m = 2$, $n = 17$, $G = 15$, $\tau_1 = 14$, $\tau_2 = 10$, $\ldots$, $\tau_{17} = 7$, and $\tau_{18} = \tau_{n+1} = G + 1 = 16$. A chain decomposition is for example given by the sequence of indices $16, 12, 11, 10, 9, 7, 5, 3, 2, 15, 8, 4, 1, 18$ with $k_1 = 9$, $k_2 = 4$, $y_1^1 = 16$, $y_9^1 = 15$, $y_1^2 = 15$, $y_4^2 = 1$, and $y_1^3 = 18$. The first sequence is $[\tau_{16}, \tau_{12}] = [1, 2]$ and just contains the car 1. The next chains are also just single cars $[\tau_{12}, \tau_{11}] = [2, 3] = (2)$, $\ldots$, $[\tau_9, \tau_7] = [5, 6] = (5)$, followed by $[\tau_7, \tau_5] = [6, 8_2] = (6, 7)$, $[\tau_5, \tau_3] = [8_2, 9] = (8_1, 8_2)$, and two more singleton chains $[\tau_3, \tau_2] = [9, 10] = (9)$ and $[\tau_2, \tau_{15}] = [10, 11] = (10)$, which cover the last cars of the grey train. The yellow cars are then covered by the chains $[\tau_{15}, \tau_8] = [11, 12] = (11)$, $[\tau_8, \tau_4] = [12, 13_1] = (12, 13_2)$, $[\tau_4, \tau_1] = [13_1, 14] = (13_1)$, and finally $[\tau_1, \tau_{18}] = [14, 16] = (14, 15)$. The length of this decomposition is $\max(k_1, k_2) = \max(9, 4) = 9$.

The dummy car τ_{n+1} is only required as a terminal in the above definition, it is not a member of any chain of the decomposition. It can also be regarded as the single car of the $(m + 1)$th train, as done in Lemma 3 below. This lemma establishes a relation between chain decompositions of inbound sequences of cars and feasible schedules. Recall from page 24 that the binary representation of an integer number k is denoted by $[k]_2$.

Lemma 3 *Let $\mathcal{T} = \tau_1, \ldots, \tau_n$, $\tau_x \in \{1, \ldots, G\}$, $x = 1, \ldots, n$, be the inbound sequence of cars for a classification instance with m outbound trains and $\tau_{n+1} := G+1$. Let further a chain decomposition be given by $y_1^1, \ldots, y_{k_1}^1, y_1^2, \ldots, y_{k_2}^2, \ \ldots \ , y_1^m, \ldots, y_{k_m}^m, y_1^{m+1}$, in which car $\tau_{y_i^j}$ belongs to the jth outbound train, $j = 1, \ldots, m$.*

Then, putting $b^{\tau_x} = [i-1]_2$ for every car τ_x of chain $[\tau_{y_i^j}, \tau_{y_{1+1}^j}]$ and $b^{\tau_x} := [k_j - 1]_2$ for every car of $[\tau_{y_{k_j}^j}, \tau_{y_1^{j+1}}]$ yields a feasible schedule of length $\lceil \log_2(k-1) \rceil$ where k denotes length of the chain decomposition.

Proof Let τ_x, τ_y be two cars of the same outbound train j. If $\tau_x = \tau_y$, i.e. τ_x and τ_y are of the same type, neither (1.2) nor (1.3) impose any relation for their assigned bitstrings b^{τ_x} and b^{τ_y}. Hence, w.l.o.g., assume $\tau_x < \tau_y$. If τ_x and τ_y are in the same chain, then $b^{\tau_x} = b^{\tau_y}$, so (1.2) is fulfilled. Since a chain is nondecreasing by Definition 1, $x < y$ holds and thus (1.3) is fulfilled as well.

Consider the case where the cars τ_x and τ_y are in different chains $\tau_x \in [\tau_{y_i^j}, \tau_{y_{i+1}^j}]$ and $\tau_y \in [\tau_{y_{i'}^j}, \tau_{y_{i'+1}^j}]$ (or $\tau_y \in [\tau_{y_{i'}^j}, \tau_{y_1^{j+1}}]$ with $i' = k_j$). Suppose that $i' < i$; then, by Definition 2, $\tau_{y_{i'}^j} \leq \tau_y \leq \tau_{y_{i'+1}^j} \leq \tau_{y_i^j} \leq \tau_x$, which contradicts $\tau_x < \tau_y$. Hence, $i < i'$ holds, so $b^{\tau_x} = [i-1]_2 < [i'-1]_2 = b^{\tau_y}$ and both (1.2) and (1.3) are satisfied also in this case. Therefore, the schedule is feasible.

If leading zeros are ignored, the longest occurring bitstring in the assignment determines the schedule length. The longest bitstring of any car in the jth outbound train, $j = 1, \ldots, m$, will be $[k_j - 1]_2$ and is assigned to the cars in $[\tau_{y_{k_j}^j}, \tau_{y_1^{j+1}}]$. In total, the longest occurring bitstring is given by $[\max_{j=1\ldots m} \{k_j - 1\}]_2$. It has a length of $\lceil \log_2(\max_{j=1\ldots m} \{k_j\} - 1) \rceil$, which presents the length of the schedule. □

This lemma essentially says that cars of the same chain can be assigned the same bitstring. For different chains the "greater" one must be assigned a greater bitstring unless it belongs to the next outbound train. Applying this rule to the chain decomposition from just above exactly yields the grey part of the schedule of Figure 1.3(l). The first chain of the yellow train may be assigned a smaller bitstring by putting $b^{11} := 0000$, and all

the bitstrings of subsequent chains can then be decreased correspondingly. Furthermore, car 15 is contained in the same chain as 14 and would thus consequently be assigned $b^{15} := b^{14} = 0100$ by the above rule. As a result of Lemma 3, the smaller the length of a chain decomposition is, i.e., the higher the degree of presortedness, the shorter a schedule is obtained. In fact, a shortest chain decomposition yields an optimal schedule, which is shown in the following section.

1.2.5 Deriving Optimal Schedules

An algorithm for deriving optimal schedules is developed in this section. More precisely, applying Lemma 3 from the previous section to a chain decomposition of minimum length will be implicitly shown to yield an optimal schedule.

The optimal decomposition is found growing the chains one after another, each one in a greedy manner. The types are considered in ascending order. When a chain is started, all cars of the smallest type with unassigned cars are assigned to this chain. (For the very first chain, these are all the cars of type "1".) Then, the type is incremented, and every car of this type that arrives later than all cars of the currently grown chain is added. If this does not hold for all cars of this type, these cars cannot be assigned to the current chain, and a new chain must be started with them. Otherwise, extending the current chain continues with the next type. A chain also finishes whenever the first type of the next outbound train is reached.

This procedure is formally described by Algorithm 1. It receives as its input the number of cars n, the inbound sequence of cars $\mathcal{T} = \tau_1, \ldots, \tau_n$, the number of outbound trains m, and their respective greatest types $G^1, \ldots, G^m$. The algorithm repeatedly iterates over the inbound sequence of cars, once for every chain. The car index is iterated in variable x, variable j contains the outbound train currently considered, i the index of the current chain w.r.t. the current train, variable t the type currently searched, and the flag *first* indicates whether this type is the smallest in the chain currently grown. The array count[ ] is there to keep track of how many cars of every type are still unassigned.

This array is initialized in lines 1 to 4 of Algorithm 1, and the type is initialized with $t = 1$ (line 5). In every iteration

Algorithm 1: Greedy Chain Decomposition

Input: no. of cars n, inbound sequence of cars $\mathcal{T} = \tau_1, \ldots, \tau_n$, no. of outbound trains m with resp. greatest types $G^1, \ldots, G^m$

Output: indices $y_1^1 \,..\, y_{k_1}^1, y_1^2 \,..\, y_{k_2}^2, \ldots, y_1^m \,..\, y_{k_m}^m, y_1^{m+1}$ presenting chain decomposition of $\mathcal{T}$

```
1  for (t = 1, ..., G) do
2    set count[t] = 0
3  for (x = 1, ..., n) do
4    count[τ_x] ++
5  set t = 1
6  for (k = 1, ..., m) do
7    set i = 0
8    while (t ≤ G^k) do
9      set first = TRUE
10     i ++
11     for (x = 1, ..., n) do
12       if (τ_x = t) then
13         count[t] --
14         if (count[t] = 0) then
15           if (first = TRUE) then
16             put y_i^k = x
17           set first = FALSE
18           t ++
19           if (t > G^k) then
20             break
21 put y_1^{m+1} = n + 1
22 return y_1^1 .. y_{k_1}^1, y_1^2 .. y_{k_2}^2, ..., y_1^m .. y_{k_m}^m, y_1^{m+1}
```

corresponding to a chain (line 8), the flag *first* is set (line 9). In the subsequent iteration over the car sequence (line 11), the counter for the current type t is repeatedly decremented until either the end of the sequence is reached or all cars of this type are assigned. In the latter case (line 14), the lower chain index y_i^j is set in line 16 if this has been the smallest type of the chain (line 15).

The iteration over the inbound sequence then proceeds with the next type $t+1$ unless this belongs to the next outbound train $j+1$, in which case the iteration of line 11 over the cars is left. Also, the iteration of line 8 over the chains of the current train will break in this case, and the algorithm proceeds with the next train in line 6. When all types are processed, the terminal of the chain decomposition is set in line 21 and the decomposition is returned. A chain decomposition computed in this way will also be called a *greedy* chain decomposition, since as many cars as possible are greedily added to the chain currently grown.

The following lemma shows that the output of Algorithm 1 actually is a chain decomposition, its proof straightforwardly verifies that all the four conditions of Definition 2 hold.

Lemma 4 *Let $\mathcal{T} = \tau_1, \ldots, \tau_n$, $\tau_x \in \{1, \ldots, G\}$, $x = 1, \ldots, n$, be the inbound sequence of cars for a classification problem with m outbound trains, and let G^j denote the greatest occurring type in the jth outbound train, $j = 1, \ldots, m$.*

Then, given n, $\mathcal{T}$, m, and $G^1, \ldots, G^m$ as the input, the output of Algorithm 1 presents a chain decomposition of $\mathcal{T}$.

Proof Let $y_1^1 \,..\, y_{k_1}^1, y_1^2 \,..\, y_{k_2}^2, \ldots, y_1^m \,..\, y_{k_m}^m, y_1^{m+1}$ be the corresponding output of Algorithm 1. Initially, $t := 1$, and this type belongs to the first outbound train. When the while-loop of line 8 is entered for the first time, *first* is set to true and i to 1. While iterating over the sequence of cars (line 11), count[1] is decremented for every car of the first type. When the last such car is met, line 14 as well as line 15 evaluate to true, so $y_1^1 = \max\{1 \le x \le n \,|\, \tau_x = 1\}$ as required in Definition 2.

Next, consider any $j \in \{1, \ldots, m\}$ and $i \in \{1, \ldots, k_j\}$. Consider the execution (meaning *all* iterations) of the while-loop of line 8 that corresponds to this value of i (in the jth iteration of loop 6). The inbound sequence of cars is traversed by the for-loop of line 11 from τ_1 up to τ_n. Since the value of t never

decreases during this traversal, the subsequence of cars for which the check of line 12 yields true is nondecreasing. In the following paragraph, this subsequence is shown to exactly match the sequence defined by $[y_i^j, y_{i+1}^j]$. As an immediate consequence, $[y_i^j, y_{i+1}^j]$ is nondecreasing and thus a chain by Definition 1.

First, at the time when y_i^j is set in line 16, the value of *first* is true, so it is the first type during this loop for which count$[t] = 0$ in line 14. Thus, no car $\tau_x < \tau_{y_i^j}$ yields true in line 12. Second, for any car τ_x of type $\tau_x = \tau_{y_i^j}$ with $x \leq y_i^j$, τ_x is considered before $\tau_{y_i^j}$ in this loop, so line 12 yields true. Third, the value of t is raised beyond $\tau_{y_i^j}$ when $\tau_{y_i^j}$ is visited, so line 12 is false for any $\tau_x = \tau_{y_i^j}$ with $x > y_i^j$. Fourth, if $\tau_{y_i^j} < \tau_x < \tau_{y_{i+1}^j}$, then count$[\tau_x] = 0$ before this iteration of the for-loop of line 11 finishes. Thus, line 13 is executed for every car of this type, so line 12 must yield true for every such car τ_x. Finally, let $\tau_x = \tau_{y_{i+1}^j}$, and define α as the greatest index of all cars of the type preceding $\tau_{y_{i+1}^j}$, i.e. $\alpha := \max_{z=1\ldots n} \{z \mid \tau_z = \tau_{y_{i+1}^j} - 1\}$. As $t = \tau_{y_{i+1}^j}$ when the while-loop finishes, t must be incremented from $t = \tau_{y_{i+1}^j} - 1$ in line 18, so count$[\tau_\alpha] = 0$ must hold just after τ_α is checked. Hence, if $x > \alpha$, then τ_x yields true when checked in line 12. At the end of the loop, the value of count$[\tau_x]$ thus corresponds to the number of cars of type τ_x with $x < \alpha$. Since count$[\tau_x] = 0$ when y_{i+1}^j is set in the subsequent execution of the loop, there is no car of type τ_x between $\tau_{y_{i+1}^j}$ and τ_α, which would be contained in $[y_i^j, y_{i+1}^j]$ according to its definition. Therefore, $[y_i^j, y_{i+1}^j]$ contains exactly those cars τ_x for which the check of line 12 is true in this execution. By the argument of the previous paragraph, $[y_i^j, y_{i+1}^j]$ presents a chain.

Similarly, consider $[y_{k_j}^j, y_1^{j+1}]$ for any $j \in \{1, \ldots, m\}$ and the execution of the while-loop of line 8 corresponding to $i = k_j$ (in the jth execution of loop 6). First, at the time $y_{k_j}^j$ is set in line 16, it is the first type with count$[t] = 0$ during this execution of the while-loop, so no car $\tau_x < \tau_{y_{k_j}^j}$ yields true in line 12. Second, for any car $\tau_x = \tau_{y_{k_j}^j}$ with $x \leq y_{k_j}^j$, τ_x is considered before $\tau_{y_{k_j}^j}$ in this loop, so line 12 yields true. Third, the value of

t is raised beyond $\tau_{y^j_{k_j}}$ when $\tau_{y^j_{k_j}}$ is visited, so line 12 is false for any $\tau_x = \tau_{y^j_{k_j}}$ with $x > y^j_{k_j}$. Fourth, in case $\tau_{y^j_{k_j}} < \tau_x < \tau_{y^{j+1}_1}$, then count$[\tau_x] = 0$ before this iteration of the for-loop is finished, so line 12 yields true for every car of this type as line 13 must be executed for all of them to get down to count$[\tau_x] = 0$. Finally, let $\tau_x = \tau_{y^{j+1}_1}$, and define $\alpha := \max_{z=1\dots n} \{z \mid \tau_z = \tau_{y^{j+1}_1} - 1\}$. Since $t = \tau_{y^{j+1}_1}$ at the end of this iteration of the while-loop, t must be incremented from $t = \tau_{y^{j+1}_1} - 1$, so count$[\tau_\alpha] = 0$ just after τ_α is checked in line 12. Hence, if $x > \alpha$, then τ_x yields true when checked in line 12, so count$[\tau_x]$ contains the number of cars of type τ_x with $x < \alpha$ at the end of this loop. Since count$[\tau_x] = 0$ when y^{j+1}_1 is set in the subsequent execution of the loop, there cannot be any car of type τ_x between $\tau_{y^{j+1}_1}$ and τ_α. Altogether, $[y^j_{k_j}, y^{j+1}_1]$ contains exactly those cars for which the check of line 12 is true in this execution. Since the value of t never decreases, also $[y^j_{k_j}, y^{j+1}_1]$ is nondecreasing and thus a chain.

Since also $y^{m+1}_1 = n+1$ by line 21, the output of Algorithm 1 satisfies all the conditions of Definition 2 and therefore is a chain decomposition. □

Regarding the asymptotic running time of Algorithm 1, initializing *count*[ ] in lines 1 to 4 is done in $\mathcal{O}(G+n)$, which equals $\mathcal{O}(n)$ as $G \leq n$. For every iteration of the for-loop in line 11, a constant number of operations is performed, and completing a loop corresponds to finishing a single chain. Hence, growing a single chain takes $\mathcal{O}(n)$ time. If $c = \sum_{j=1}^{m}(k_j + 1)$ denotes the number of chains computed by the algorithm, growing all the chains is done in $\mathcal{O}(cn)$ time, which also presents the total running time. This yields a time of $\mathcal{O}(n^2)$ independently of the output because $c \leq n$.

As an immediate consequence of Lemma 4, the application of Lemma 3 to the output of Algorithm 1 yields a feasible schedule. This can be done during the execution of the algorithm by assigning a car just after its type is matched in line 12. This corresponds to adding an additional line to the code of Algorithm 1 as follows:

```
  ⋮
12          if (τ_x = t) then
12a           b^{τ_x} := [i − 1]_2
13            count[t] − −
  ⋮
```

Applied to the inbound sequence of Figure 1.3(a), this yields a schedule that corresponds to Figure 1.3(l) for all cars of the grey train. The first car of the yellow train, i.e. car 11, would be assigned $b^{11} = 0000$ instead of $b^{11} = 0001$, i.e. the next smaller bitstring, just as described at the end of Section 1.2.4. Also the remaining cars would get the respective next smaller bitstring, and car 15 would again be assigned the same bitstring as car 14 since both are in the same chain.

This method is finally shown to be optimal in the following theorem.

Theorem 2 *Let $\mathcal{T} = \tau_1, \ldots, \tau_n$, $\tau_x \in \{1, \ldots, G\}$, $x = 1, \ldots, n$, be the inbound sequence of cars for a classification problem with m outbound trains, and let $\tau_{n+1} := G+1$.*

Then, given n, $\mathcal{T}$, m, and $G^1, \ldots, G^m$ as the input, Algorithm 1 computes an optimal schedule.

Proof Let B be the schedule computed by Algorithm 1 and h denote its length. Assume, for contradiction, there is a shorter feasible schedule. Among all shorter feasible schedules, let B^* denote the one that, w.r.t. ordering (1.4), maximizes the smallest car s_k for which B and B^* differ, i.e., $b^{*s_k} \neq b^{s_k}$. Recall the bitstrings of B^* can be assumed to be nondecreasing w.r.t. ordering (1.4) by Lemma 2. In case $b^{*s_k} > b^{s_k}$, schedule B^* remains feasible when putting $b^{*s_k} := b^{s_k}$ and will not increase in length. This contradicts the maximality assumption above.

For the remainder of the proof, thus assume $b^{*s_k} < b^{s_k}$. Let $[t_i, t'_j]$ with $t < t'$ be the chain computed by Algorithm 1 that contains car s_k. The cars s_k and t_i are in the same chain, so $b^{s_k} = b^{t_i}$. There are four cases. Firstly, let $t_i < s_k$ w.r.t. (1.4), i.e., either $t < s$ or $s = t$ and $k < i$. Then, $b^{t_i} = b^{*t_i}$ by the maximality assumption above. However, B^* is nondecreasing, so $b^{*s_k} \geq b^{*t_i}$. The assumption $b^{s_k} > b^{*s_k}$ thus yields $b^{*t_i} = b^{t_i} = b^{s_k} > b^{*s_k} \geq b^{*t_i}$, which is a contradic-

tion. Secondly, let $s = t > 1$ and $k = i$. Consider the latest car of type $t-1$ in the chain $[\cdot, t_i]$ preceding chain $[t_i, t'_j]$, i.e. car $(t-1)_\alpha$ with $\alpha = \max_{z-1\ldots n} \{z \,|\, (t-1)_z \in [\cdot, t_i]\}$. This car satisfies $b^{*(t-1)_\alpha} = b^{(t-1)_\alpha}$ by the maximality assumption and $b^{(t-1)_\alpha} = b^{t_i} - 1$ by the way the algorithm assigns bitstrings. Assumption $b^{*t_i} < b^{t_i}$ thus yields $b^{*t_i} \leq b^{*(t-1)_\alpha}$. As t_i is not assigned to the chain of $(t-1)_\alpha$ by the greedy algorithm, it must arrive before $(t-1)_\alpha$ in the inbound sequence of cars. This, however, contradicts the feasibility constraint (1.3) for B^*. Thirdly, if $t = s = 1$, then s_k is contained in the first chain of the algorithm's output, so $b^{s_k} = [0]_2 = 0$ and $b^{*s_k} \not< b^{s_k}$.

Therefore, there is no case in which schedule B^* may exist, so Algorithm 1 computes a schedule of minimum length. □

Since Algorithm 1 yields an optimal schedule, the underlying chain decomposition has minimum length by Lemma 3. This lemma therefore yields the optimal length value $h^* = \lceil \log_2 k^* \rceil$ with k^* denoting the length of a shortest chain decomposition.

The schedule shown in Figure 1.3(l) differs from the one found by Algorithm 1 as mentioned before. Still, with its four steps it presents an optimal schedule too. Observe that its length is determined by the partial schedule for the grey train here, and there are many more feasible schedules of length four for this example.

The proofs of Lemma 2 and particularly Lemma 3 are based on conditions (1.2) and (1.3) being sufficient for feasibility. If further restrictions apply, such as the limited length of classification tracks mentioned in Section 1.1.4, however, this assumption does not pertain any longer, and the chain decomposition calculated by Algorithm 1 may not correspond to a feasible schedule. The optimal schedule length $h^* = \lceil \log_2 k^* \rceil$ just established thus only presents a (tight) lower bound on the length of feasible schedules in general. The following chapter deals with the important restriction of sorting tracks having a limited capacity, which was mentioned in Section 1.1.4. Before, the four most commonly applied sorting methods are revisited in the following section.

1.2.6 Traditional Methods Revisited

Using the schedule encoding introduced above, a more efficient description of the traditional multistage methods is achieved. As announced in Section 1.1.3, four different methods are analyzed in this section: sorting by train and (basic) simultaneous sorting, as well as the two simultaneous sorting variants triangular and geometric sorting. The presentation includes the aspect of presorted inbound trains.

All schedules that follow a common classification method share common characteristics. For the example of triangular sorting—an explanation of which follows below—every car is rolled in at most three times, so every schedule that accurately implements the triangular method for some problem instance has this property. Every classification schedule can be represented by the encoding, which itself reflects the characteristics of the schedule and thus the characteristics of the method the schedule implements. Therefore, every classification method can be regarded as a *class of classification schedules* with certain characteristics and thus corresponds to a *class of encodings*.

The descriptions all assume that m outbound trains are given, with respective numbers of cars $n_1, \ldots, n_m$ and numbers of groups $g_1, \ldots, g_m$ with a fixed order in their respective outbound trains. Further, define $g_{\max} := \max_{1 \le k \le m} g_k$ and $g_{\min} := \min_{1 \le k \le m} g_k$. In terms of types, even though the concept is not required for the following presentations, these assumptions say that the first outbound train contains all cars of type $t \in \{1, \ldots, g_1\}$, the second all cars of type $t \in \{g_1{+}1, \ldots, g_1{+}g_2\}$, etc.

Sorting by Train

The basic multistage method of sorting by train [Dag86, Kre62, Sid72] comprises two stages. At their initial roll-in, inbound cars are separated according to their outbound trains by sending all cars of a common outbound train to the same classification track. The resulting unordered outbound trains are processed successively in the second stage: a train is pulled back over the hump and rolled in again, sorting the cars according to their groups by sending the cars of each group to the same track.

The groups of cars are moved from the tracks in the required order and coupled to form the sorted outbound train, and the process continues with the next train.

The corresponding class of encodings is given as follows: for the encoding $b_{h-1}^{\tau_x} \dots b_0^{\tau_x}$ of any car τ_x in the jth group of the kth outbound train, $b_i^{\tau_x} = 1$ only for $i = k - 1 + \sum_{k'=1}^{k-1} g_{k'}$ (corresponding to the initial roll-in) and $i = k-1+\sum_{k'=1}^{k-1} g_{k'} + j$ (corresponding to the second stage).

Note that the roll-ins of the second phase do not consider presortedness, so exactly one track is used for each group. The length h of this schedule is given by the number of pull-outs $h = m + \sum_{k=1}^{m} g_k$, which does not cover assembling the groups into trains after the second stage. This method occupies exactly m classification tracks after the first stage, so the total number of required tracks is at least $m+g_{\min}-1$ and at most $m+g_{\max}-1$. The latter number is tight if the first train processed in the second stage has $g_{\max}$ groups.

Sorting by train is sometimes called *initial grouping according to outbound trains* [Sid72]. The corresponding names used in the German literature are *Staffelverfahren* and *Ordnungsgruppenverfahren* [Kre62]

Simultaneous Sorting

The first stage of the two-stage method called simultaneous sorting [Fla53, Kre62, Pen59, Sid72] sorts according to the cars' groups in the outbound trains in contrast to sorting by train. Hence, a track contains cars of several outbound trains after the first stage in general, but only from one group of each train as presortedness is disregarded. In the encoding of a schedule of this class, this step forces $b_{j-1} = 1$ for every car of the jth group, $j = 1, \dots, g_k$, of the kth outbound train, $k = 1, \dots, m$. In the second stage, the cars are sorted according to their outbound trains: the tracks are successively pulled in the order of the corresponding group index, and each set of cars pulled out is directly rolled back in, always sending cars of a common outbound train to the same output track. As mentioned before, this last roll-in is not explicitly given in the encoding.

Since presortedness is not considered by this method, there is one pull-out for every group after the first stage, so simultaneous

sorting yields a number $g_{\max}$ of sorting steps. This number is maximal among all variants of simultaneous sorting for an unrestricted classification yard but still lower than for sorting by train. Regarding the track requirement, exactly $g_{\max}$ tracks are used in the first stage. Thus, at most $g_{\max} + m - 1$ tracks are needed since up to $m-1$ further tracks are needed as output tracks, and at least $g_{\min} + m - 1$ tracks are required: even after shifting all group indices of all trains such that the jth group in the kth train becomes the $(j + g_{\max} - g_k + 1)$th group of the kth train, $k = 1, \ldots, m$, $j = 1, \ldots, g_k$, pulling the first of the last $g_{\min}$ tracks to be pulled in the second stage forces starting to make all m outbound trains (if not yet started), so exactly $g_{\min} + m - 1$ tracks are occupied at this point.

Observe that in the above description cars are never guided to an output track during primary sorting. This may be desired for operational reasons and is even necessary if the output tracks cannot be accessed from the primary hump such as for the layout of Figure 1.2(b). If all output tracks can be accessed from the primary hump, however, the schedule becomes one step shorter, whereas the required number of tracks does not change.

This is also done by Keckeisen [Kec58], who gives an early example for integrating presortedness of inbound trains into the basic form of simultaneous sorting. The number of steps in this example is reduced from four to two, while the track requirement reduces from nine to seven. Though this is a significant saving, the track requirement is not reduced consequently as one more track could be saved for the example in [Kec58].

In contrast to sorting by train, assembling all outbound trains is performed simultaneously, which explains the name of the method. Simultaneous sorting is also called the *simultaneous method*, *simultaneous marshalling* [Pen59], *sorting by block* [Dag86], or *initial grouping according to subscript* [Sid72]. The corresponding German terms are *Simultanverfahren* and furthermore *Elementarsystem* [Kre62] when explicitly referring to the basic version just described.

In this basic form of simultaneous sorting, every car is pulled out once and rolled in twice, once in either stage. For the following variants this restriction is dropped. Instead of stages, these variants are specified by sequences of sorting steps.

Triangular Sorting

Triangular sorting is considered in [Pen59, Kre62, Sid72, DDH83, Dag86]. This method may be regarded as a superclass of simultaneous sorting that is given by allowing at most three roll-ins for each car (including its final roll-in to an output track). For the schedule encoding, this yields the restriction of $b_i = 1$ for at most two of the indices $i = 0, \ldots, h-1$ in the bitstring of every classified car.

For this classification method Krell [Kre62] gives an upper bound of $\frac{1}{2}h(h+1)$ on the maximum number of groups $g_{\max}$ of the outbound trains that can be sorted in h steps. The description does not consider any presortedness of inbound trains, and the given bound is only tight if the cars of an outbound train with $g_{\max}$ groups arrive in completely reversed order in the inbound sequence. Hence, if this result is extended with the concept of chains, a better bound is achieved in general. Let $k_1, \ldots, k_m$ denote the respective numbers of chains in a chain decomposition according to Definition 2 with length k. The number of distinct bitstrings $b_{h-1} \ldots b_0$ of length h and $b_i = 1$ for at most two different indices $i \in \{0, \ldots, h-1\}$ is given by $\binom{h}{1} + \binom{h}{2} = \binom{h+1}{2}$. These $\binom{h+1}{2}$ suitable bitstrings can of course be ordered according to (1.1). With this, an assignment similar to that of Lemma 3 may be applied: instead of putting $b^{\tau_x} := [i-1]_2$ for every car τ_x of the ith chain of the jth train, τ_x is assigned the ith smallest of the suitable bitstrings. The argument of the proof of Lemma 3 still applies here to show (1.2) and (1.3) are satisfied in the resulting schedule. Thus, classifying by triangular sorting can be done within h sorting steps if $k \leq \binom{h+1}{2} = \frac{1}{2}h(h+1)$. Conversely, if for some outbound train the number of chains is k, a set of k bitstrings must be chosen with $b_i = 1$ for at most two different $i \in \{0, \ldots, h-1\}$. This yields a feasible schedule of length $h = \lceil \sqrt{2k} - \frac{1}{2} \rceil$. The method can be generalized to any bound on the number of roll-ins for a car.

The triangular-like occupation of the classification tracks after the initial roll-in accounts for the name of this variant. In [Kre62], triangular sorting is called *Vorwärtssortierung bei höchstens zweimaligem Ablauf.*

Geometric Sorting

The method of *geometric sorting* [Fla53, Pen59, Kre62, Sid72] derives from simultaneous sorting for dropping the restriction on the number of roll-ins completely. This corresponds to bitstrings with no restriction at all. The performance of this method is given in the literature by $g_{\max} \leq 2^h - 1$ for h sorting steps [Kre62] if the order of the cars in the inbound sequence is disregarded, which is suboptimal in general. Notice the difference of this number from 2^h by one for cars not to be sent to the output track in the initial roll-in. This corresponds to disallowing the all-zero bitstring in the bitstring representation. For Algorithm 1 this would mean replacing assignment $b^{\tau_x} := [i-1]_2$ in line 12a by $b^{\tau_x} := [i]_2$. If the inbound order is considered, applying the concept of chains to this method turns out to exactly yield the class of schedules computed by Algorithm 1, with a bound of $k \leq 2^h$ (or $k \leq 2^h - 1$), where k is the length of the greedy chain decomposition.

Considering the special case of a single outbound train of $2^j - 1$ groups for some integer $j > 0$, the initial roll-in sends 2^{j-i} groups to the ith track, $i = 1, \ldots, k$; the sum of these numbers gives the geometric sum, which explains the method's name. Geometric sorting is called *maximale Vorwärtssortierung* in [Kre62].

As mentioned before, geometric sorting minimizes the schedule length assuming the inbound order of cars is completely unknown. Also, the capacities of tracks must be sufficiently large—if this cannot be assumed, the sorting methods of the next chapter should be considered. Before, the setting with a flexible order of trains is studied in the following section.

1.2.7 Optimizing the Order of Inbound Trains

In the preceding sections the order of inbound trains was given by their time of arrival at the classification yard as mentioned in Section 1.2. After its arrival a train is brought to one of the parallel tracks of the receiving yard. From there the trains are successively brought to the lead track, which may be done in an order different from the one they arrived at the yard. Since the number of tracks in the receiving yard is in general much smaller than the number of inbound trains, this order cannot be

chosen completely freely. Still, the question arises what is the order of inbound trains that minimizes the number of sorting steps. Finding this order presents an $\mathcal{NP}$-hard problem, which is shown in the following. This will be done by a reduction from the problem of finding an order of inbound trains that yields the shortest greedy chain decomposition, to which an $\mathcal{NP}$-hard feedback set problem can be reduced.

The considerations of Section 1.2.5 imply that, in order to minimize the optimal schedule length, it suffices to find the permutation of inbound trains that minimizes the length of the greedy chain decomposition computed by Algorithm 1. For the case of unique cars, the algorithm finds chains by greedily growing maximal strictly ascending sequences of consecutive cars. The end of a chain is reached whenever the next greater car arrives earlier than the thereby last car of the chain. The following definition introduces a notion for such pairs of consecutive cars in reversed order.

Definition 3 *Let $\mathcal{T} = \tau_1, \ldots, \tau_n$ be a permutation of the sequence $(1, \ldots, n)$, and let $\mathcal{T}$ be the inbound sequence of cars for a given classification instance with m outbound trains.*

For any car τ_x, $x \in \{1, \ldots, n-1\}$, the unique pair (τ_x, τ_y), $y \in \{1, \ldots, n\}$, with $\tau_y = \tau_x + 1$, will be called a break *of $\mathcal{T}$ if $x > y$ and τ_y is in the same outbound train as τ_x. If further τ_x and τ_y belong to the same inbound train, (τ_x, τ_y) will be called an* internal *break; otherwise, it will be called an* external *break.*

In a greedy chain decomposition for the case of unique cars and a single outbound train, a pair of cars presents a break if and only if they are the last car of a chain and the first car of the subsequent chain. Hence, the number of breaks in an inbound sequence of cars determines the length of its greedy chain decomposition and thus the length of the corresponding schedule. More important here actually is the converse: finding an order that yields an optimal schedule can be applied to find an order that minimizes the number of breaks, which is shown in the following lemma.

Lemma 5 *Let $T_1, \ldots, T_\ell$ be ℓ inbound trains with a total number of n unique cars in a classification problem with a single outbound train.*

Finding an order of inbound trains that minimizes the number of breaks can be reduced in polynomial time to finding an order of inbound trains that minimizes the length of an optimal classification schedule.

Proof Let T_j denote the inbound train containing the car n. Consider the following modification of the problem instance: for some nonnegative integer k, successively add cars $n+1$, $n+2, \ldots, n+k$ at the front of T_j, and call the result *k-modification* of the original instance. This introduces exactly k additional internal breaks, which are there for every order of inbound trains since the number of internal breaks is independent of the order of trains. Now, let x denote the number of (internal and external) breaks for any order of trains that yields an optimal schedule for $k = 0$, i.e. for the original instance. If h^* denotes the length of this schedule, then $2^{h^*-1} \leq x \leq 2^{h^*} - 1$. Note that $2^{h^*} \leq n$ here.

For any $k \geq 0$, let $h(k)$ denote the length of an optimal schedule for an order of trains that minimizes the optimal schedule length in the k-modification. Successively increment k from $k = 0$ until the value k is found for which $h(k) = h^*$ but $h(k+1) = h^* + 1$. With $h(k) = h^*$ and the considerations of Section 1.2.5, the corresponding order induces at most $2^{h^*} - 1$ breaks in the k-modification, which means at most $2^{h^*} - 1 - k$ breaks in the original instance. Assume, for contradiction, there is an order inducing less than $2^{h^*} - 1 - k$ breaks in the original instance. This order induces less than $(2^{h^*} - 1 - k) - (k+1) = 2^{h^*}$ breaks in the $(k+1)$-modification with a corresponding minimum schedule length of at most h^* steps. This is a contradiction to $h(k+1) = h^* + 1$, so the assumption is wrong, i.e., there is no order inducing less than $2^{h^*} - 1 - k$ breaks in the original instance. Therefore, the order corresponding to the above value k with $h(k) = h^*$ yields exactly $2^{h^*} - 1 - k$ breaks in the original problem instance and this number is minimal.

Regarding the computational complexity, constructing a k-modification can be done in $\mathcal{O}(k) \subseteq \mathcal{O}\left(2^{h^*}\right) \subseteq \mathcal{O}(n)$, and the number of calculations of values of $h(\cdot)$ is $\mathcal{O}\left(2^{h^*-1} - 1\right) \subseteq \mathcal{O}(n)$. This yields a total of $\mathcal{O}\left(n^2 p(n)\right)$ if the computation time of $h(n)$ is $\mathcal{O}(p(n))$. □

A certain $\mathcal{NP}$-hard feedback set problem can further be reduced to minimizing the number of breaks as shown in the

following. Consider the minimum feedback arc set (MFAS) problem for general directed (multi-)graphs (see e.g. [FPR09]): given a directed (multi-)graph $G = (V, E)$, find an ordering $\pi : V \to \{1, \ldots, \ell\}$, $\ell = |V|$, of its vertices that minimizes the number of edges $(u, v) \in E$ for which $\pi(u) > \pi(v)$. Such an edge is called a *backward edge* in the ordering. The following lemma relates the MFAS problem for a special graph class to the problem of finding an optimal order of inbound trains.

Lemma 6 *For the class of directed multigraphs the edges of which present an Euler path, the MFAS problem can be reduced in polynomial time to the problem of finding an order of inbound trains that minimizes the number of external breaks w.r.t. a single common outbound train.*

Proof Let $P = (v_1, \ldots, v_n)$, $v_k \in V$, $k = 1, \ldots, n$, denote an Euler path of a given multigraph $G = (V, E)$ with $\ell = |V|$. For each node $v \in V$, introduce an inbound train T_v. Initialize train T_{v_1} with the single car 1. Walk along P and add for every visited edge (v_k, v_{k+1}), $k = 1, \ldots, n-1$, the car $k+1$ to the front of train $T_{v_{k+1}}$. This train classification instance can be constructed in $\mathcal{O}(|V| + |E|)$ linear time. Note that this construction yields n unique cars.

Let $\pi : V \to \{1, \ldots, \ell\}$ be a linear ordering of V, and consider the sequence of trains $\mathcal{T} = T_{\pi^{-1}(1)}, \ldots, T_{\pi^{-1}(\ell)}$. If an edge (v_k, v_{k+1}) of P is a backward edge in π, then $\pi(v_k) > \pi(v_{k+1})$, so train T_{v_k} occurs after $T_{v_{k+1}}$ in $\mathcal{T}$. As T_{v_k} contains k and $T_{v_{k+1}}$ contains $k+1$, $(k, k+1)$ is an external break. If an edge (v_k, v_{k+1}) of P is not a backward edge in π, then $\pi(v_k) < \pi(v_{k+1})$, so k arrives before $k+1$ and the pair $(k, k+1)$ is not a break.

Summarizing, the construction yields a one-to-one correspondence between linear orderings of V and permutations of trains for which the respective number of backward edges corresponds to the number of external breaks. Therefore, the minimum number of external breaks yields the minimum number of backward edges. □

The graph class used in Lemma 6 is a superclass of Euler graphs. The MFAS problem for Euler graphs can be shown to be $\mathcal{NP}$-hard [FG10], so the problem of minimizing the number of external breaks and thus the number of sorting steps is $\mathcal{NP}$-hard as well. This is summarized in the following theorem.

Theorem 3 *Let $T_1, \ldots, T_\ell$ be ℓ given inbound trains with a total of n unique cars in a classification problem instance with a single outbound train.*

Finding an order of inbound trains that minimizes the length of an optimal classification schedule is an $\mathcal{NP}$-hard problem.

Note that the construction of Lemma 6 yields a classification instance with unique cars as mentioned in the proof, which is not affected by Lemma 5. Hence, even the subproblem without types is $\mathcal{NP}$-hard. Moreover, both lemmas consider a single outbound train, so the combination with this specialization is $\mathcal{NP}$-hard too.

1.3 Summary

The practical problem of multistage train classification has been introduced to the reader in Section 1.1, including the basic operations of rolling in and pulling out and the influence of different classification yard layouts on the sorting process. The overview of state-of-the-art methods in Section 1.1.3 has shown that there has not been done very much theoretical work on the train classification problem. This particularly holds for the aspect of presortedness, i.e. the inbound order of cars, which yields a lot of room for improvement.

A novel representation of train classification schedules has been developed, successfully applied to derive length-optimal multistage classification schedules by considering the order of inbound cars, and finally used to analyze the efficiency of commonly used methods in Section 1.2.6. Moreover, an $\mathcal{NP}$-hard optimization problem has been shown to be obtained in Section 1.2.7 if the order of inbound trains is subject to modification in a generalized setting.

Chapter Notes The main results of Section 1.2 were published in [JMMN07] in a different fashion and for distinct cars without types. Figures 1.1 and 1.2 come from [JMMN11], in which additionally the generalization to types of cars is sketched. This has now been supplemented by Section 1.2 with actual formal proofs, additionally including the general case of multiple outbound trains throughout.

Future Directions Some interesting questions arise in the interplay between primary and secondary sorting. It is not always a priori clear which traffic should go into secondary sorting. This decision depends on the available capacity of the classification yard, the distribution of work between different classification yards, and on the connected routing of blocks through a network of yards.

Moreover, it might be interesting to study the problem for other sorting requirements than the practically most relevant one presented in Section 1.1.3. A rich variety of order requirements for cars in outbound trains can be found in the theoretical paper [DMMR00] and also in [HZ07].

Chapter 2

Train Classification for Limited Track Capacities

This chapter deals with the important restriction of limited sorting track capacities. The precise setting is described in Section 2.1, followed by a review of the related work in Section 2.2. For this problem setting an exact integer programming (IP) model is presented in Section 2.3, and three variants of an approximation algorithm are described in Section 2.4. All these approaches are then analyzed and compared in the extensive experimental evaluation of Section 2.5.

2.1 Setting and Further Notation

As mentioned in Section 1.2.2 above, this chapter returns to the assumption that the order of inbound trains is fixed. Furthermore, the setting includes the case of multiple outbound trains and unique cars w.r.t. their types.

Regarding the infrastructure, every sorting track involved in the classification process is pulled exactly once here. The track pulled in the ith classification step, $i = 1, \ldots, h$, corresponding to column b_{i-1}, will be called the $(i{-}1)$*th track*. The output track will also be called the *hth track*. If there are several outbound trains, there will be one output track for every train. Still, all

these tracks will together be referred to as the hth track since it is clear which car goes to which outbound train and thus to which output track.

Most importantly, the sorting tracks are assumed to have the same finite capacity in this chapter. With this restriction the optimal schedule length derived in Section 1.2.5 only presents a lower bound. Recall the capacity of a track, denoted by C, has been defined by the highest number of cars the track may accommodate. Further, for a given schedule B, the number of roll-ins in the ith step has been called the weight of column b_i and denoted by $w(b_i)$. Hence, for any schedule B of length h derived in this section, the weight $w(b_i)$ of no column must exceed the track capacity C, i.e., $w(b_i) \leq C$ for all $i = 0, \ldots, h-1$. This will be implemented by constraint (2.2) below.

2.2 Related Work

The train classification problem for the setting of this chapter has been shown to be an $\mathcal{NP}$-hard problem in [JMMN11], even for the special case of a single outbound train. The further work related to restricted classification track capacities is restricted to some earlier papers from the field of railway engineering.

The total track capacity requirement, i.e. the sum over the single classification track capacities, of sorting by train is compared to that of the basic simultaneous method in [Kre63]. For a fixed exemplary length of output and sorting tracks, a formula depending on the number of outbound trains and the number and variance of their groups is given. With regard to the applicability of different classification methods, the yard in Münster, Germany, is mentioned in [End63] as an example where the simultaneous method cannot directly be applied due to the yard's small track capacities. In an attempt to relate put-through times of cars in single-stage sorting to the physical characteristics of a classification yard, Petersen gives formulas for the probability of yard congestions resulting from limited classification track capacities in [Pet77a, Pet77b]. Formulas for the average number of cars in a yard are derived by Daganzo in [Dag86, Dag87a] in order to estimate the average total track length requirement for different multistage methods in yard design. For single examples of traffic in combination with sorting

by train and the triangular method, Daganzo also sketches how the total track requirement and number of tracks can be related to the required lengths of individual classification tracks. Some other railway publications, such as [Boo57, Pen59, Sid72], just mention the fact that the classification tracks must have a certain length for the respective sorting method to be applicable. However, no alternatives are provided for the case when a certain method fails due to unfulfilled capacity requirements.

2.3 Optimal Integer Programming Approach

This section provides an integer programming model for the train classification problem with limited track capacities, which presents an $\mathcal{NP}$-hard problem as mentioned in the previous section. This IP model applies the binary encoding of classification schedules of Section 1.2.3 to implement the feasibility conditions of Theorem 1 and the requirements w.r.t. the sorting track capacities.

Following the bitstring notation of Section 1.2.3 for a schedule B of length h, there are binary variables b_i^τ, $i = 0, \ldots, h-1$, $\tau = 1, \ldots, n$, corresponding to car τ in the ith sorting step for a classification problem with n cars. Let $F \subseteq \{1, \ldots, n\}$ denote the subset of cars that are the first of their respective outbound trains. Let further $\mathrm{rev}\colon \{1, \ldots, n\} \times \{1, \ldots, n\} \to \{0, 1\}$ be an indicator function with $\mathrm{rev}(\tau, \sigma) = 1$ if and only if the cars τ and σ arrive in reversed order in the inbound sequence of cars. Note that $\mathrm{rev}(\tau, \sigma) = \mathrm{rev}(\sigma, \tau)$. With these definitions, consider the following integer programming model:

$$\min \sum_{i=1}^{h} \sum_{\tau=1}^{n} b_i^\tau$$

$$\text{s.t.} \sum_{i=0}^{h-1} 2^i b_i^\tau \geq \mathrm{rev}(\tau, \tau-1) + \sum_{i=0}^{h-1} 2^i b_i^{\tau-1}, \ \tau \in \{1 \,..\, n\} \setminus F \quad (2.1)$$

$$\sum_{\tau=1}^{n} b_i^\tau \leq C, \quad i \in \{0, \ldots, h-1\} \quad (2.2)$$

$$b_i^\tau \in \{0, 1\}, \quad i \in \{0, \ldots, h-1\}, \ \tau \in \{1, \ldots, n\} \quad (2.3)$$

Note that the sum $\sum_{i=0}^{h-1} 2^i b_i^\tau$ gives the integer represented by the bitstring b^τ. Constraints (2.1) enforce a feasible schedule w.r.t. the ordering of cars in the outbound trains according to Theorem 1: if two consecutive cars $\tau-1$ and τ of the same outbound train are in correct order in the inbound train, they may either be assigned the same bitstring or a greater bitstring is assigned to τ in order to satisfy (1.2); otherwise, $\mathrm{rev}(\tau-1,\tau)=1$, so τ will get a strictly greater bitstring than $\tau-1$ as required according to (1.3). Observe that Theorem 1 does not impose any conditions for pairs of cars $\tau-1,\tau$ of different outbound trains, which is why pairs with $\tau \in F$ are excluded from condition (2.1). The sum $\sum_{\tau=1}^{n} b_i^\tau$ matches the definition of the weight of column b_i, so constraints (2.2) implement the restricted capacities of the sorting tracks.

This integer programming model contains the length h of the solution as a parameter, and its objective function minimizes the total number of cars rolled in during the classification process, i.e. the weight of the schedule. This presents only a secondary objective as mentioned in Section 1.1.4. In order to minimize the primary objective, i.e. the number of sorting steps, a sequence of integer programs can be solved with increasing length values h. This is explained in more detail in Section 2.5.1.

2.4 Algorithmic Approximations

This section deals with algorithmic approaches for the $\mathcal{NP}$-hard train classification problem with limited track capacities. A basic 2-approximation is introduced in Section 2.4.1, for which an optimality condition is given in Section 2.4.2, followed by heuristic improvements of the algorithm in Section 2.4.3.

2.4.1 Basic 2-Approximation

The approximation algorithm for restricted track capacities comprises two consecutive steps: first, a schedule is derived that does not necessarily satisfy the capacity constraints (2.2) but the relaxed constraint (2.4) introduced below. Second, every column that violates (2.2) in this schedule is distributed over several newly introduced columns which all meet their capacity constraints.

Consider the following constraint on the weight $w(B)$ of a schedule B of length h, i.e. on the number of bits b_i^τ in B with $b_i^\tau = 1$:

$$w(B) \leq hC \tag{2.4}$$

If a schedule satisfies the capacity constraints (2.2), it will satisfy (2.4) too. The proof of the following lemma shows how an optimal schedule w.r.t. (2.4) can be found efficiently, which presents the first part of the approximation algorithm. It is formulated for a single outbound train, the generalization to multiple outbound trains is given right after. Note that the order requirements for outbound trains defined in Section 1.2.2 result in an outbound train of the form $U = 1, \ldots, n$ here.

Lemma 7 *Given a classification problem with an inbound sequence $\mathcal{T}$ of n cars to be classified into a single outbound train, a minimum-length feasible schedule $\tilde{B}$ satisfying (2.4) can be obtained in polynomial time.*

Proof Consider any subsequence $u = (\tau, \tau+1, \ldots, \tau+k-1)$, $1 \leq \tau$, $\tau + k - 1 \leq n$, of k consecutive cars of the outbound train $U = 1, \ldots, n$. During the execution of any classification schedule, if u is located on some classification track, it must have got there by sending some (possibly empty) suffix $u'' = (\tau + j, \ldots, \tau + k - 1)$, $0 \leq j \leq k$, of u to the track while the corresponding prefix $u' = (\tau, \ldots, \tau + j - 1)$ had been standing there. There are $k+1$ possible suffixes, so u can be produced in $k + 1$ different ways. The outbound train U corresponds to the subsequence for $\tau = 1$ and $k = n$.

This idea results in the following dynamic programming approach for computing a minimum-length schedule satisfying (2.4) that additionally minimizes the weight for this length. An entry $T(i, u)$ of the dynamic programming table denotes the minimum weight of a schedule that produces u within i steps on any track. Its value is determined by ranging over all possible suffixes u'' of u if $i > 0$:

$$T(i, u) = \min_{u'u''=u} \left(T(i-1, u') + T(i-1, u'') + |u''|\right) \tag{2.5}$$

In this recurrence formula, $u'u''$ denotes the concatenation of u' and u'' and $|u''|$ the length of u''. The table is initialized for $i = 0$ with $T(0, u) = |u|$ if u occurs as a subsequence in

the inbound sequence of cars $\mathcal{T}$. This corresponds to the initial roll-in. Otherwise, $T(0,u) = \infty$. Then, the table is filled "row-wise" using formula (2.5), where each row corresponds to a single value of i, while columns correspond to subsequences. The row index is increased until an entry $T(\tilde{h}, U)$ in the column of U meets $T(\tilde{h}, U) - n \leq \tilde{h}C$. This value $\tilde{h}$ is bounded by n. Recall that the final roll-in to the output track is included in the value $T(\tilde{h}, U)$ but not in condition (2.4). Hence, $T(\tilde{h}, U)$ presents the optimal solution value and $\tilde{h}$ the number of sorting steps.

The corresponding schedule $\tilde{B}$ of length $\tilde{h}$ is recursively reconstructed starting from this entry $T(\tilde{h}, U)$. For any entry $T(i,u)$ considered during the reconstruction, the bits $\tilde{b}^{\tau}_{i-1}$ of $\tilde{B}$ will be determined for all cars τ in the subsequence u as follows: let $T(i-1,u')$ and $T(i-1,u'')$ be the entries that $T(i,u)$ has been derived from. Now, put $\tilde{b}^{\tau}_{i-1} = 1$ for every car τ of u'' as u'' came from the $(i-1)$th track in the ith step, and put $\tilde{b}^{\tau}_{i-1} = 0$ for every car τ of u' as u' had been standing on the track where u is built in the ith step. Then, recurse with this construction for $T(i-1,u')$ and $T(i-1,u'')$ unless $i = 0$.

The resulting schedule $\tilde{B}$ is feasible: since the algorithm only concatenates subsequences u' and u'' of consecutive cars but never splits them, for every pair τ, $\tau+1$ of consecutive cars there is a unique step i in which they go to the same track for the first time. After this they take exactly the same course over the classification tracks. Let $T(u,i)$ be a corresponding entry during the reconstruction with τ and $\tau+1$ in u, but τ in u' and $\tau+1$ in u''. Then, $\tilde{b}^{\tau}_{h-1}...\tilde{b}^{\tau}_{i} = \tilde{b}^{\tau+1}_{h-1}...\tilde{b}^{\tau+1}_{i}$, $\tilde{b}^{\tau}_{i-1} = 0$, and $\tilde{b}^{\tau+1}_{i-1} = 1$, so $\tilde{b}^{\tau} < \tilde{b}^{\tau+1}$. As a special case, τ and $\tau+1$ arrive in correct order in the inbound sequence of cars $\mathcal{T}$; they will then be reconstructed the same bitstring. Together, the reconstructed schedule is feasible by Theorem 1.

A single entry $T(i,u)$ can be calculated in time $\mathcal{O}(|u|)$, so the whole table can be computed in $\mathcal{O}\left(\tilde{h}n^3\right)$ since the number of subsequences of consecutive cars, i.e. the width of the table, is given by $\binom{n}{2} \in \mathcal{O}\left(n^2\right)$. The reconstruction time for the schedule $\tilde{B}$ is linear in its encoding size, which is $\mathcal{O}\left(\tilde{h}n\right)$. Since $\tilde{h} \leq n$, $\tilde{B}$ can be derived in polynomial time. □

Note that the dynamic programming algorithm in the above proof minimizes the weight of the schedule for each length value.

This is only required to prove the minimum length w.r.t. (2.4) of the final schedule. The approximation's second stage, which is described further below, works with a minimum-length w.r.t. (2.4) of *any* weight.

The algorithm in the proof of Lemma 7 can be extended to the case of multiple outbound trains $U_1, \ldots, U_m$. There is one dynamic programming table T_k for every outbound train U_k, and these tables are computed independently of each other. Still, their computations must be performed in parallel by increasing the value of i, i.e. the row index or number of classification steps, simultaneously for all tables in each step. Each step i thus results in m schedules $\tilde{B}_1, \ldots, \tilde{B}_m$ of the same length i, and (2.4) is checked to hold for their "vertical" concatenation:

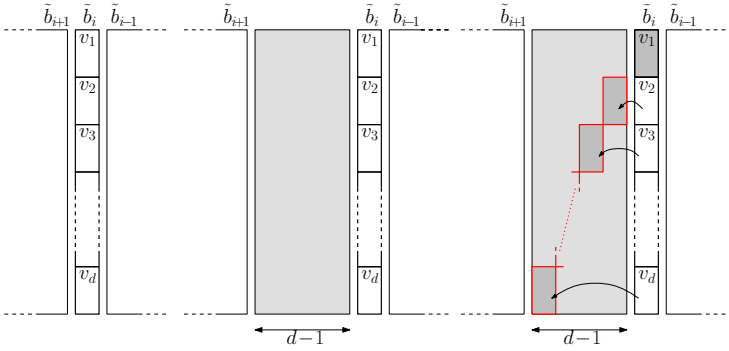
$$B' = \begin{pmatrix} \tilde{B}_1 \\ \vdots \\ \tilde{B}_m \end{pmatrix} \tag{2.6}$$

More precisely, the row index i is incremented until the condition $\sum_{k=1}^{m} T_k(i, U_k) \leq iC$ is met. A single table entry is still derived only from entries of the same table, and the schedule reconstruction can be done independently for every outbound train. This finishes the first stage of the approximation algorithm. A schedule produced by this stage will be referred to as an *intermediate* schedule, and it will serve as the input for the second stage described in the following.

Let $\tilde{B}$ be an intermediate schedule obtained by Lemma 7—for either case of a single or multiple outbound trains—, and let $\tilde{h}$ denote its length. If B^* is an optimal schedule satisfying (2.2), B^* will also satisfy (2.4) as mentioned above. Therefore, the length h^* of B^* meets $h^* \geq \tilde{h}$. This fact will be used in the following theorem which finally states the 2-approximation. Its proof directly yields an iterative algorithm to derive the corresponding schedule from the intermediate schedule $\tilde{B}$.

Theorem 4 *For any instance of the train classification problem with restricted track capacities that has an optimal solution of length h^*, a feasible schedule of length at most $2h^*$ can be obtained in polynomial time.*

Proof Let $\tilde{B}$ be a feasible schedule of minimum length $\tilde{h}$ that fulfils the relaxed constraint (2.4). This schedule is transformed



(a) $\tilde{b}_i$ violating (2.2) (b) $d-1$ columns are added (c) $\tilde{b}_i$ is distributed

Figure 2.1: A single iteration of the second part (corresponding to Theorem 4) of the approximation algorithm for bounded capacities, applied to a column $\tilde{b}_i$ that violates the capacity constraints (2.2).

iteratively, performing one iteration step for each column of $\tilde{B}$ that violates the capacity constraints (2.2). The steps can be performed in arbitrary order. Figure 2.1 depicts a single iteration step, which works as follows: let $\tilde{b}_i$ be a column of $\tilde{B}$ with weight $w(\tilde{b}_i) > C$. Define $d := \left\lceil \frac{w(\tilde{b}_i)}{C} \right\rceil$ and divide $\tilde{b}_i$ into d consecutive segments $v_1, \ldots, v_d$ such that each segment v_k with $k < d$ has a weight $w(v_k) = C$ and v_d carries the remaining weight, which is at most C (Figure 2.1(a)). Now, $d-1$ additional zero-columns are inserted between $\tilde{b}_i$ and $\tilde{b}_{i+1}$ (Figure 2.1(b)). Every v_k is moved to the left by $k-1$ columns, overwriting the corresponding segment of the zero-column, and the moved entries in $\tilde{b}_i$ are set to zero (Figure 2.1(c)). This finishes the iteration step for $\tilde{b}_i$.

Let b^τ and $b^{\tau+1}$ be two bitstrings of the schedule B under transformation just before an iteration step corresponding to some column b_i, and let h denote the current length. If τ and $\tau+1$ are in different outbound trains, they are not subject to any order requirements. W.l.o.g. assume that τ and $\tau+1$ belong to the same outbound train. If both b_i^τ and $b_i^{\tau+1}$ belong to the same segment v_k, the cars τ and $\tau+1$ will retain the order relation for their bitstrings as both bitstrings are extended in the same way. Otherwise, b_i^τ is contained in some

segment v_k and $\tau+1$ in the successor segment v_{k+1}. If then $b^\tau = b^{\tau+1}$, the bitstrings read as follows after the iteration: $b^\tau = \tilde{b}^\tau_{h-1}...b^\tau_{i+1}0..01..0b^\tau_i...b^\tau_0 < \tilde{b}^\tau_{h-1}...b^\tau_{i+1}0..10..0b^\tau_i...b^\tau_0 = b^{\tau+1}$. However, if the two bitstrings satisfy $b^\tau < b^{\tau+1}$ already before the iteration, either $\tilde{b}^\tau_{h-1}...b^\tau_{i+1} < \tilde{b}^{\tau+1}_{h-1}...b^{\tau+1}_{i+1}$ holds with inserting and changing bits right of column b_{i+1} having no effect on their order constraints, or $\tilde{b}^\tau_{h-1}...b^\tau_{i+1} = \tilde{b}^{\tau+1}_{h-1}...b^{\tau+1}_{i+1}$, in which case $\tilde{b}^\tau_{h-1}...b^\tau_{i+1}0..01 < \tilde{b}^{\tau+1}_{h-1}...b^{\tau+1}_{i+1}0..10$. In either case $b^\tau < b^{\tau+1}$ after the iteration too. Therefore, each iteration preserves the order constraints of the schedule, so the final schedule is feasible. Moreover, after each step, the respective fixed column meets its capacity constraint (2.2), which also holds for every newly introduced column, so the final schedule satisfies (2.2).

For every column in $\tilde{B}$, this procedure yields at most one column with weight less than C in the final schedule. Furthermore, the final number of columns with weight equal to C will not exceed $\tilde{h}$ as $\tilde{B}$ satisfied (2.4), so the final length h will not exceed $2\tilde{h}$. Since $\tilde{h} \leq h^*$, inequality $h \leq 2h^*$ holds.

Inserting all the columns with weight C can be done in $\mathcal{O}(\tilde{h}n)$, inserting all columns with lower weight in $\mathcal{O}(\tilde{h}n)$, and rewriting the original columns in $\mathcal{O}(\tilde{h}n)$ as well. The intermediate schedule $\tilde{B}$ can be derived in $\mathcal{O}(\tilde{h}n^3)$ by Lemma 7, which dominates the polynomial total running time of $\mathcal{O}(\tilde{h}n^3)$. □

The basic approximation algorithm that is derived in the proofs of Lemma 7 and Theorem 4 will be referred to by BASE. Some variants of BASE will be presented in Section 2.4.3.

2.4.2 A Condition for Optimality

The just presented basic approximation algorithm always yields an intermediate schedule $\tilde{B}$ that presents an optimal solution w.r.t. the relaxed constraint (2.4). It may happen that this schedule is not extended in the second stage of the algorithm because all of its columns meet the capacity constraints. In this case $\tilde{B}$ presents an optimal schedule w.r.t. the capacity constraints (2.2) as shown in the following theorem.

Theorem 5 *For any train classification problem, if a feasible schedule $\tilde{B}$ of minimum length satisfying (2.4) also satisfies (2.2), then $\tilde{B}$ is an optimal schedule satisfying (2.2).*

Proof Let B^* be an optimal schedule satisfying (2.2) and h^* be its length. Since $h \leq h^*$ holds for every feasible schedule B of length h that fulfils (2.4), $\tilde{h} \leq h^*$ with $\tilde{h}$ denoting the length of $\tilde{B}$. Further, as $\tilde{B}$ satisfies (2.2), also $\tilde{h} \geq h^*$, so $\tilde{h} = h^*$ and $\tilde{B}$ is an optimal schedule satisfying (2.2). □

In other words, if the approximation algorithm does not extend an intermediate schedule, it will yield an optimal schedule. Since a satisfied condition (2.4), however, does not generally imply condition (2.2) to hold, Theorem 5 only presents a sufficient but no necessary condition for optimality.

2.4.3 Heuristic Improvements

In the basic approximation algorithm BASE described in Section 2.4.1, there are m partial schedules $\tilde{B}_1, \ldots, \tilde{B}_m$ of length $\tilde{h}$ at the end of the first stage, where $\tilde{B}_k$ denotes the partial schedule covering all the cars of outbound train U_k, $k = 1, \ldots, m$. It might happen that the dynamic programming table of the kth outbound train obtains some feasible schedule B'_k with weight $w(B'_k) = w(\tilde{B}_k)$ in an earlier iteration corresponding to some length value $h'_k < \tilde{h}$. Constraint (2.4) may not be satisfied yet at this point. One reason for this might be relatively high weights of the current partial schedules of other outbound trains for this length value h'_k. Then, the basic algorithm continues with appending a leading zero-column $b_{i-1} = \mathbf{0}$ to B'_k in all subsequent steps $i = h'_k + 1, \ldots, \tilde{h}$. The actual length of $\tilde{B}_k$ is thus $h'_k < \tilde{h}$. Besides, (partial) schedules are reconstructed in Lemma 7 in a way not to yield any zero-columns which are not leading. In this way, schedules derived by BASE tend to have many columns of low index violating their respective capacity constraints (2.2), while the weight of columns remains far behind C for higher indices.

In order to overcome this, two improvements of BASE are developed in the following. The algorithms, which will be called SHIFT and INS, try to level the occupation of the sorting tracks at the end of the first stage. The improved algorithms are based on the following theorem.

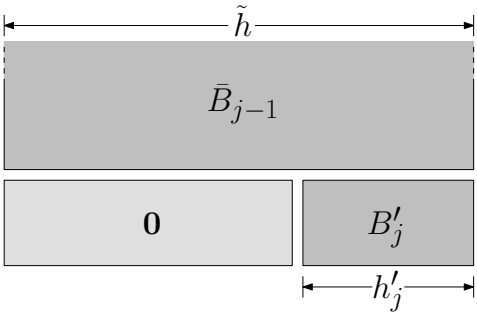
Theorem 6 *For any feasible schedule $B = b_{h-1} \ldots b_0$ of length h, inserting a zero-column $b_j = \mathbf{0}$ at any position $j \in \{0, \ldots, h\}$ of B yields a feasible schedule of length $h + 1$.*

Proof Let $b^\tau = b^\tau_{h-1} \dots b^\tau_0$ and $b^\sigma = b^\sigma_{h-1} \dots b^\sigma_0$ be the two bitstrings of B for two cars τ and σ. If $b^\tau = b^\sigma$, then clearly $b^\tau_{h-1} \dots b^\tau_j 0 b^\tau_{j-1} \dots b^\tau_0 = b^\sigma_{h-1} \dots b^\sigma_j 0 b^\sigma_{j-1} \dots b^\sigma_0$ too, so inserting an all-zero column does thus not violate the feasibility conditions of Theorem 1 between τ and σ in this case.

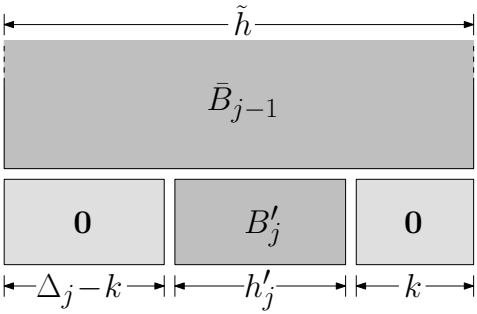
If $b^\tau < b^\sigma$, i.e. $\sum_{i=0}^{h-1} 2^i b^\tau_i < \sum_{i=0}^{h-1} 2^i b^\sigma_i$ by ordering (1.1), then ineqality $\sum_{i=0}^{j-1} 2^i b^\tau_i + \sum_{i=j}^{h-1} 2^{i+1} b^\tau_i < \sum_{i=0}^{j-1} 2^i b^\sigma_i + \sum_{i=j}^{h-1} 2^{i+1} b^\sigma_i$ holds for all $j \in \{0, \dots, h\}$. Hence, bitstring $b^\tau_{h-1} \dots b^\tau_j 0 b^\tau_{j-1} \dots b^\tau_0$ is strictly smaller than $b^\sigma_{h-1} \dots b^\sigma_j 0 b^\sigma_{j-1} \dots b^\sigma_0$ for all $j \in \{0, \dots, h\}$. Inserting an all-zero column does thus not violate feasibility of B. □

As a consequence of the above theorem, distributing $\tilde{h} - h'_k$ zero-columns between any pairs of columns of B'_k yields a feasible partial schedule of length $\tilde{h}$. Put differently, if any h'_k positions are chosen to which to distribute the columns of B'_k without changing their relative order and then zero-columns are assigned to the remaining $\tilde{h} - h'_k$ positions, a feasible schedule of length $\tilde{h}$ is obtained. This fact is now made use of to derive two further variants of the basic approximation algorithm, called SHIFT and INS, that effectively reduce the number and scale of capacity violations. Both variants still apply the dynamic program of Lemma 7 like BASE but without adding leading zero-columns. Instead, two heuristics are applied that distribute the columns of partial schedules with a small actual length in more sophisticated ways, which is described below. The second stage follows Theorem 4 again for all BASE, SHIFT, and INS equally.

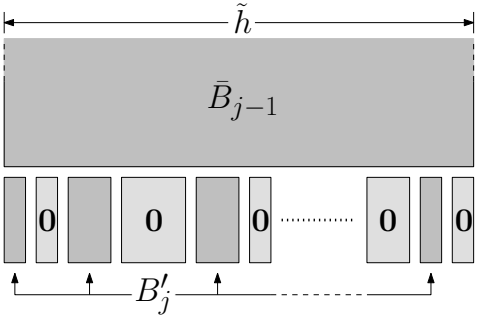
At the end of the first stage, SHIFT and INS first proceed in common ways and sort the partial schedules that correspond to the outbound trains in decreasing order according to their *actual* lengths. W.l.o.g. let $B'_1, \dots, B'_m$ be the ordered partial schedules with actual length values $h'_1 \geq \dots \geq h'_m$. Both variants start with an empty schedule $\bar{B}_0$ of length $\tilde{h}$ and successively append the partial schedules to obtain concatenations $\bar{B}_j$, $j = 1, \dots, m$, similar to the picture shown in (2.6). SHIFT and INS take different approaches to perform the concatenations, which is described in the following two paragraphs and illustrated in Figure 2.2. After the last concatenation in any variant, $\bar{B}_m$ finally presents the intermediate schedule $\tilde{B}$ satisfying (2.4), which is then passed to the second stage.



(a) BASE



(b) SHIFT



(c) INS

Figure 2.2: Three variants of the approximation algorithm obtained by three different ways of combining the partial schedules of outbound trains to a common intermediate schedule at the end of the first stage.

Consider the jth step of concatenation for the approximation variant SHIFT, $j = 1, \ldots, m$. Define $\Delta_j := \tilde{h} - h'_j$, which corresponds to the number of columns the partial schedule $\bar{B}_j$ remains behind the intermediate length $\tilde{h}$. According to Theorem 6, for any value $k = 0, \ldots, \Delta_j$, adding k consecutive trailing zero-columns to B'_j and $\Delta_j - k$ consecutive leading zero-columns yields a feasible partial schedule of length $\tilde{h}$. This schedule can be appended to $\bar{B}_{j-1}$ to obtain $\bar{B}_j$ as the next step towards $\tilde{B}$. This step is illustrated in Figure 2.2(b). Now, SHIFT obtains $\bar{B}_j$ by choosing that parameter value $k \in \{0, \ldots, \Delta_j\}$ which minimizes the value of $\sum_{i=0}^{\tilde{h}-1} (C - (w(b_i) \bmod C))$, where b_i is the ith column *after* appending B'_j to $\bar{B}_{j-1}$. This objective is motivated as follows: if $w(b_i) > C$ for some column i in the concatenation, an additional column will certainly be introduced in the second stage according to Theorem 4. Hence, at this point this column can be filled further just below a weight of $2C$, which then does not trigger introducing any further columns in the second stage. This difference in weight, i.e. the number of 1-bits that can be added to b_i until the next multiple of C is reached, is precisely given by $(C - (w(b_i) \bmod C))$. The objective of SHIFT thus tries to use up big gaps in columns to subsequent multiples of C before exceeding such a multiple in another column. In contrast, BASE implicitly puts $k = 0$ in every concatenation step, which yields the picture shown in Figure 2.2(a). Note that every schedule computed by BASE may also be achieved by SHIFT in general.

The approximation variant INS has even more flexibility in concatenating the partial schedules compared to SHIFT. Consider the jth concatenation step of INS, $j = 1, \ldots, m$. The algorithm determines the value of expression $(w(b_i) \bmod C)$ for all columns b_i of $\bar{B}_{j-1}$, $i = 0, \ldots, \tilde{h} - 1$, where here b_i is the ith column of $\bar{B}_{j-1}$ before concatenating. Then, the single columns of the partial schedule B'_j are concatenated to those h'_j columns of $\bar{B}_{j-1}$ that have the lowest values of $(w(b_i) \bmod C)$. The relative order of the columns of B'_j is not changed. To the remaining $\tilde{h} - h'_j$ columns of $\bar{B}_{j-1}$, zero-columns are appended following Theorem 6, which yields $\bar{B}_j$ for the next iteration as illustrated in Figure 2.2(c). This choice of the h'_j columns is motivated by trying to use up gaps between subsequent multiples of C

instance	n	ℓ	m	$n_{\max}$	k	k_{tot}
MON	486	49	24	47	4	52
TUE	329	44	24	30	5	48
WED	310	47	24	31	4	49
THU	364	44	24	46	4	49
WKD	368	43	27	41	4	52

Table 2.1: The five real-world instances corresponding to five days of traffic and their parameters in detail: n (no. of cars), ℓ (no. of inbound trains), m (no. of outbound trains), $n_{\max}$: length of longest outbound train, k: length of greedy chain decomposition, k_{tot}: total no. of chains.

similarly to the objective of SHIFT described above. INS concatenates more flexibly than SHIFT, but every schedule obtained by SHIFT may still be constructed by INS as well. Both algorithms are compared to each other and to further approaches in the experimental study of the next section.

2.5 Experimental Evaluation

In this section the three algorithmic variants of the previous section are compared to each other and to the integer programming approach of Section 2.3 in an extensive experimental evaluation, using various real-world and synthetic train classification instances. The design of the experiments, including the test instances, is described in the subsection below, their outcome follows in Section 2.5.2.

2.5.1 Experimental Setup

This section describes the train classification instances for the evaluation, the precise way in which the experiments were conducted, and the objectives by which their outcome is assessed.

Test Instances

The traffic data of a whole week in 2005 for the Swiss classification yard Lausanne Triage was provided by courtesy of

Stephan Leber from SBB Infrastructure. From this data the five instances summarized in Table 2.1 were extracted which cover all the cars handled by multistage sorting. The instances have between 43 and 49 inbound trains with total numbers of cars ranging from 310 to 486 and between 24 and 27 outbound trains. The order of the inbound trains is fixed according the arrival times of the trains at the yard. In order to obtain unique cars, as required by the 2-approximation of the previous section, cars of the same type were converted to distinct types as follows: if $y_1^1, \ldots, y_{k_1}^1, y_1^2, \ldots, y_{k_2}^2, \ \ldots \ , y_1^m, \ldots, y_{k_m}^m, y_1^{m+1}$ denotes the greedy chain decomposition of a classification instance with m outbound trains, the sequence of cars comprising the first chain $[\tau_{y_1^1}, \tau_{y_2^1}]$ were converted into the sequence of consecutive cars $(1, \ldots, c_1)$, where c_1 denotes the length of the chain; chain $[\tau_{y_2^1}, \tau_{y_3^1}]$ was converted to $(c_1+1, \ldots, c_2)$, where c_1 is the chain's length; etc. In other words, whenever two cars of the same type appeared successively in the greedy decomposition, the type of the second car and that of all subsequent cars was incremented. The greedy chain decompositions of the five instances all had a length of four or five as shown in Table 2.1.

Note that an instance comprises traffic data independent of any yard infrastructure, particularly without any capacity value. The term "problem" will refer to a pair of an instance and a capacity value. Lausanne Triage has a track capacity of about $C = 40$, and the five instances were combined with similar track capacities $C \in \{30, \ldots, 50\}$, which yielded 105 real-world classification problems in total.

In order to test the approaches of Section 2.3 and 2.4 for more complex train classification problems, synthetic instances were derived in the style of the above real instances. As the first crucial parameter of a derived problem, the number of cars n was fixed to one of three values $n \in \{200, 400, 800\}$. Then, the lengths of the outbound trains were drawn uniformly at random from one of the intervals $[2 \ldots 20]$, $[2 \ldots 40]$, or $[2 \ldots 60]$ until the sum of all lengths exceeded n, at which value the last outbound train was cut off. The breaks w.r.t. the outgoing trains were geometrically distributed with probability $p = \frac{c-m}{n}$, where the parameter c took a value $c \in \{30, 60, 120\}$ if $n = 200$, $c \in \{60, 120, 240\}$ if $n = 400$, and $c \in \{120, 240, 480\}$ if $n = 800$. Note that m denotes the actual number of outbound trains,

instances	n	length distribution	c
$1, \ldots, 4$	200	$[2 \ldots 20]$	30
$5, \ldots, 8$			60
$9, \ldots, 12$			120
$13, \ldots, 16$		$[2 \ldots 40]$	30
$17, \ldots, 20$			60
$21, \ldots, 24$			120
$25, \ldots, 28$		$[2 \ldots 60]$	30
$29, \ldots, 32$			60
$33, \ldots, 36$			120
$37, \ldots, 40$	400	$[2 \ldots 20]$	60
$41, \ldots, 44$			120
$45, \ldots, 48$			240
$49, \ldots, 52$		$[2 \ldots 40]$	60
$53, \ldots, 56$			120
$57, \ldots, 60$			240
$61, \ldots, 64$		$[2 \ldots 60]$	60
$65, \ldots, 68$			120
$69, \ldots, 72$			240
$73, \ldots, 76$	800	$[2 \ldots 20]$	120
$77, \ldots, 80$			240
$81, \ldots, 84$			480
$85, \ldots, 88$		$[2 \ldots 40]$	120
$89, \ldots, 92$			240
$93, \ldots, 96$			480
$97, \ldots, 100$		$[2 \ldots 60]$	120
$101, \ldots, 104$			240
$105, \ldots, 108$			480

Table 2.2: Four synthetic instances for each combination of three parameter values: n (no. of cars), interval of uniform distribution of outbound train lengths, c (parameter in geometric distribution of breaks with $p = \frac{c-m}{n}$).

which had been obtained in the previous step of the construction. With four instances for every combination of parameters, there were 108 instances in total as summarized in Table 2.2. For the integer programming of Section 2.3, each such problem instance was combined with capacity values $C \in \{10, 20, \ldots, 60\}$, yielding 648 synthetic classification problems. Their faster running times compared to the integer programming approach allowed evaluating the algorithms of Section 2.4 for a bigger set of problems, and the instances were combined with capacities $C \in \{10, 20, 30, \ldots, 10\lfloor \frac{n}{30} \rfloor\}$, yielding 1'620 synthetic instances for the approximation algorithms.

Schedule Computation

As mentioned before, the integer programming model of Section 2.3 contains the schedule length h as a parameter and minimizes the weight of the solution schedule in its objective function. In order to minimize the primary objective h, a short sequence of IP models was solved with increasing values for h. The first model passed to the solver corresponded to a fixed value of $h = 1$. If this turned out to be infeasible, the value of h was successively incremented until a feasible solution was obtained. In order to temporally limit the experiments, the number of iterations had been limited to $h = 8$, and the time limit for each iteration step was set to one hour.

For this approach there are four possible outcomes for each sequence of computations. If a feasible schedule is found in some iteration, let B be the computed schedule and h denote its length. First, if B is proved optimal w.r.t. the IP objective for the model of the hth iteration and the same problem has been shown infeasible for the previous iteration with $h - 1$, then B will be called *weight-optimal*. Second, if there is no feasible solution for $h - 1$ but B has not been proved optimal by the solver, then B will be called *length-optimal*. As a third possibility, the IP solver may find a feasible schedule for a number of sorting steps h but not for $h - 1$ within the time limit. Finally, no solution is found if the eighth iteration ends with a timeout or infeasibility. As reported in more detail in Section 2.5.2 below, weight-optimal schedules were found for all real-world problems in combination with every capacity value, and the longest such schedule had a length of $h = 8$. All IP computations presented

in Section 2.5.2 were performed with ILOG Studio 3.7 featuring CPLEX 9.0 on an Intel Xeon CPU with 2.80 GHz and 2 GB main memory.

For every classification problem a schedule was computed with each BASE, SHIFT, and INS. These were compared to each other and—wherever possible—to the integer programming schedules in Section 2.5.2. In the evaluation, the shortest of the schedules returned by BASE, SHIFT, and INS for a single problem will be referred to by MIN. MIN can be regarded as a fourth algorithmic variant that runs BASE, SHIFT, and INS and then returns the minimum of the three solutions. Moreover, the four approximation algorithms were compared to a quite simple heuristic called HEUR, which is explained in detail in the next paragraph. All the approximation algorithms and HEUR were implemented in C++, compiled with the GNU compiler g++ 4.1.2, and run on an Intel Pentium IV CPU with 2.80 GHz and 1.5 GB RAM.

In order to achieve their approximation factor, all variants first derive an intermediate schedule that is optimal w.r.t. the relaxed constraint (2.4). In contrast, HEUR first computes a schedule according to Algorithm 1 without regard to (2.4). This yields an intermediate schedule $\tilde{B}$ of length $\tilde{h} = \lceil \log_2 k^* \rceil$, where k^* denotes the length of the greedy chain decomposition, which does not necessarily satisfy the capacity constraints (2.2). Then, HEUR executes its second stage exactly in the same way as the approximation algorithms: Theorem 4 is applied to $\tilde{B}$ to obtain a schedule satisfying (2.2). The approach of HEUR does not guarantee the approximation factor but has a very simple implementation. With regard to the experimental evaluation of Section 2.5.2, applying the more involved approximation algorithms can thus only be justified if they perform significantly better than HEUR.

Measuring Objectives

The length values of the approximate and heuristic schedules clearly present the most important performance measure. In particular, these values are to be compared to the lengths of weight- or length-optimal IP schedules if available. In order to characterize how far from optimal a schedule is, the notion of *relative excess* relates the length of a schedule to the optimal length: let B be a feasible schedule of length h that satisfies

	BASE	SHIFT	INS	MIN	HEUR
average excess	0.302	0.058	0.119	0.051	0.217

Table 2.3: Average relative excess of the different algorithmic approaches for the real-world problems.

(2.2), and let h^* denote the length of a shortest feasible schedule satisfying (2.2); then, the *relative excess* $\eta(B)$ of B is defined as $\eta(B) := \frac{h-h^*}{h^*}$. Note that by this definition every optimal schedule B^* satisfies $\eta(B^*) = 0$.

In order to assess derived schedules for problems for which the optimal solution is unknown, an upper bound on the relative excess can be established for approximate schedules: let $\tilde{B}$ be an optimal schedule w.r.t. (2.4) of length $\tilde{h}$; if B is a feasible schedule that satisfies (2.2), then the fraction $\ell(B) := \frac{h-\tilde{h}}{\tilde{h}}$ will be called the *relative extension* of B. Since $h^* \geq \tilde{h}$ and $h^* \leq h$, every feasible schedule B satisfies $0 \leq \eta(B) \leq \ell(B)$, so a schedule's extension bounds its excess.

In particular, if $\ell(B) = 0$, then $\eta(B) = 0$, so B is optimal. This case exactly corresponds to the optimality condition of Theorem 5 above: equation $\ell(B) = 0$ holds if and only if $h = \tilde{h}$, which itself holds if and only if the intermediate schedule was not extended in the second stage of the approximation algorithm. All approximate and heuristic solutions were checked for $\ell(B) = 0$ as reported in Section 2.5.2. In addition, the overall running times of the different approaches are reported.

2.5.2 Experimental Results

The experimental results reported in this section were obtained as explained in Section 2.5 above. First of all, the actual length values are compared for the IP approach and the different algorithms.

Length Values

Figure 2.3 shows the distributions and averages of the measured length values for all approaches on the real-world instances. As shortly mentioned in Section 2.5.1, the integer programming approach found weight-optimal schedules for all the real-world

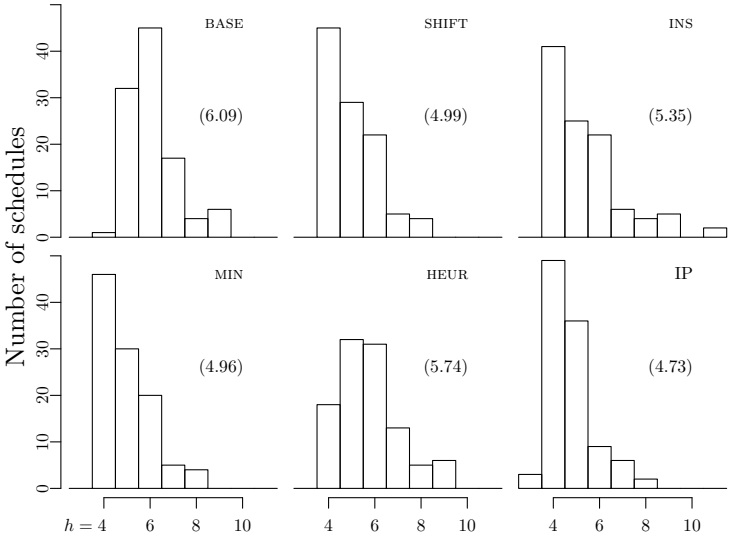


Figure 2.3: Length distributions of the real-world schedules produced by the different approximation variants, HEUR, and the IP approach. Mean values are given in parentheses.

instances, and Table 2.3 compares the algorithmic solutions via their average excess to the optimum solutions. Among all tested algorithms, BASE performed worst here with an average length of 6.09 and was even beaten by HEUR with an average of 5.74 sorting steps, which corresponds to average excesses of 30.2 % and 21.7 %, respectively, over the optimum. However, the improved approximation algorithms SHIFT and INS performed both considerably better than HEUR: the schedules produced by INS were on average 13.1 % longer than the optimum, those produced by SHIFT even only 5.4 %. Of the 105 real-world classification problems, SHIFT yielded the best result of the three approximation algorithms in 102 cases, so SHIFT and MIN almost yield the same picture in Figure 2.3. The approximate schedule of shortest length, i.e. the one computed by MIN, was on average only 5.1 % longer than the optimal IP schedule.

Table 2.4 counts the number of real-world instances for which BASE yields shorter and longer schedules compared to each of the

BASE	<		=		>	
IP	0	(0.0 %)	6	(5.7 %)	99	(94.3 %)
SHIFT	1	(1.0 %)	14	(13.3 %)	90	(85.7 %)
INS	12	(11.4 %)	18	(17.1 %)	75	(71.4 %)
MIN	0	(0.0 %)	13	(12.4 %)	92	(87.6 %)
HEUR	14	(13.3 %)	41	(39.0 %)	50	(47.6 %)

Table 2.4: Numbers and percentages of real-world classification problems for which BASE produced a shorter (<), equally long (=), or longer (>) schedule compared to the other approaches.

HEUR	<		=		>	
IP	0	(0.0 %)	26	(24.8 %)	79	(75.2 %)
BASE	50	(47.6 %)	41	(39.0 %)	14	(13.3 %)
SHIFT	3	(2.9 %)	39	(37.1 %)	63	(60.0 %)
INS	7	(6.7 %)	55	(52.4 %)	43	(41.0 %)
MIN	1	(1.0 %)	40	(38.1 %)	64	(61.0 %)

Table 2.5: Numbers and percentages of real-world problems for which HEUR yielded a schedule that was shorter (<), equally long (=), or longer (>) than that of the other approaches.

other approaches: BASE produced a shorter schedule than INS for only twelve of the instances, and SHIFT was beaten by BASE in one case only. All these numbers show that BASE is not on par even with the simple heuristic algorithm HEUR, which emphasizes the demand for enhancement. Both variants SHIFT and INS indeed improved the basic algorithm, and the considerably shorter schedules show that the ideas behind the improvements were actually fruitful.

Comparing HEUR to the other approaches for real-world instances in Table 2.5 does not yield a much better picture than BASE. HEUR may have yielded schedules of minimum length in at least 26 cases and shorter schedules than BASE for even 50 problems. Even so, HEUR beat SHIFT and INS only three and seven times, respectively, and MIN was beaten by HEUR in just one case.

Surprisingly, the seemingly more sophisticated INS did not surpass SHIFT, neither for the mean lengths shown in Figure 2.3,

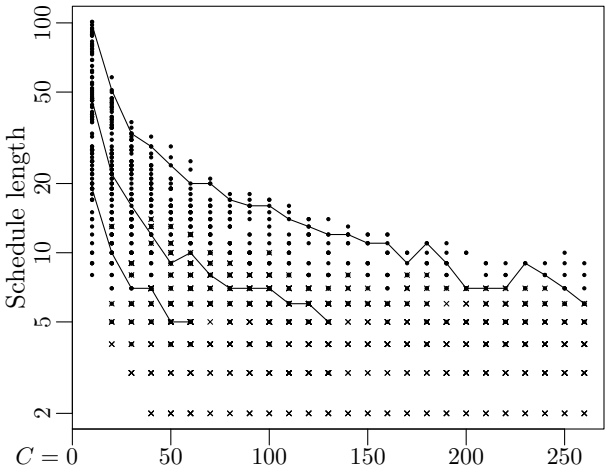


Figure 2.4: Relation of capacities of the synthetic problems and schedule lengths produced by SHIFT. ×: provably optimal schedules ($\ell = 0$), •: all other schedules ($\ell > 0$). For three instances with car numbers $n = 200$, 400, and 800, respectively, their individual data points are connected by a line for each instance.

nor for the average excess of Tables 2.3. The potential strength of INS lies in its freedom to distribute the single columns of partial schedules over non-contiguous positions when assembling the intermediate schedule from the partial schedules of outbound trains. When selecting such a set of columns in a single concatenation step, INS has to rely on the column weights before concatenation as explained in Section 2.4.3. Apparently, its flexibility does not give INS any advantage over SHIFT with its capability to take into account *resulting* column weights for its selection, even though it must be contiguous.

For the example of SHIFT, Figure 2.4 plots the computed lengths for the different synthetic instances against the capacity value of the respective instance. Schedules produced by other approximation variants show a similar trend. If a feasible schedule for a specific classification problem satisfies the capacity constraints (2.2) for some capacity C, it clearly satisfies them for the combination of the problem with every $C' > C$ as well. Therefore, the number of sorting steps h should be non-increasing for

HEUR	<		=		>	
BASE	130	(8.0 %)	314	(19.4 %)	1'176	(72.6 %)
SHIFT	8	(0.5 %)	118	(7.3 %)	1'494	(92.2 %)
INS	60	(3.7 %)	128	(7.9 %)	1'432	(88.4 %)
MIN	1	(0.1 %)	111	(6.9 %)	1'508	(93.1 %)

Table 2.6: Numbers and percentages of synthetic problems for which HEUR yielded a schedule that was shorter (<), equally long (=), or longer (>) than that of the approximation variants.

fixing a classification problem and raising its capacity C. This is mostly the case for the synthetic instances, but not always: the lines plotted in Figure 2.4 repeatedly increase, so it would be beneficial to reuse a schedule that was actually calculated for a lower capacity instead of computing a fresh one.

For the synthetic instances, feasible schedules were found for only 245 of the 648 classification problems with the IP approach, of which 165 were weight-optimal and 49 length-optimal. With a number of 153, the majority of feasible schedules was found for small problem instances with $n = 200$. Because of the limited number of iterations and the time limit (Section 2.5.1), no IP solutions were obtained for a big number of problems, for which reason directly comparing the length values of IP and approximate schedules would not be very meaningful. Instead, the algorithms are compared in terms of their relative extensions for the synthetic instances in the following section.

Extension Values

The distributions and averages of the measured relative extensions for the algorithmic approaches are shown in Figure 2.5 for the synthetic problems. With an average of 0.28, the relative extensions of BASE schedules was approximately the same as for the real-world schedules. In contrast, HEUR performed significantly worse than BASE here with a mean extension of 0.56.The extensions of schedules produced by SHIFT and INS were half as large as for BASE, and MIN still performed slightly better. A similar picture is given by Table 2.6, in which the numbers of synthetic problems are listed for which HEUR yielded a shorter, longer, or equally long schedule compared to the other algo-

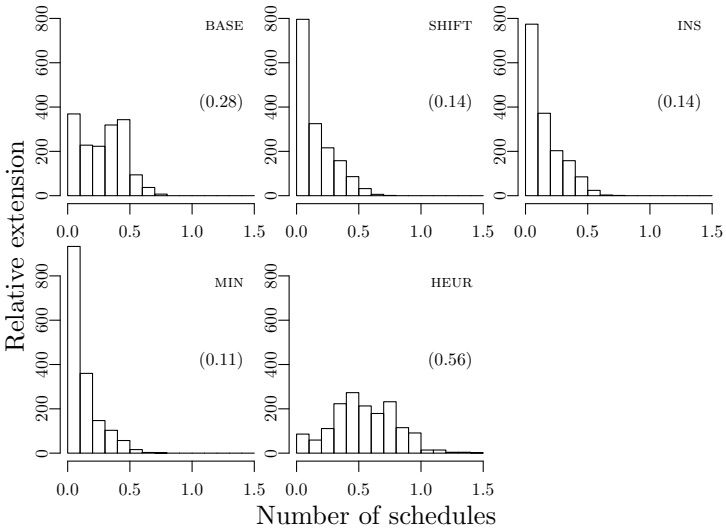


Figure 2.5: Distributions of extension values for the schedules of the synthetic problems generated by the different approximation algorithms. Mean values in parentheses.

rithms: INS was beaten by HEUR in 3.7 % of the cases, SHIFT in 0.5 %, and MIN in a single case only. All in all, SHIFT and INS improved on BASE significantly for the synthetic problems as well, and HEUR was almost completely ruled out by MIN.

As explained in Section 2.5.1, if a schedule B has a relative extension $\ell(B) = 0$, it satisfies the optimality condition of Section 2.4.2. Hence, condition $\ell(B) = 0$ presents a sufficient criterion for B to be optimal w.r.t. the capacity constraints (2.2), which can be tested if the optimal length is not available. Table 2.7 lists the numbers of problems for which this criterion was met for the individual algorithms. SHIFT produced zero-extension schedules the most frequently, followed by INS, both for the real-world and the synthetic classification problems. All the approximations variants together, which corresponds to applying MIN, found zero-extension schedules for more than three quarters of all real-world problems, whereas this was only the case for half of the synthetic problems. The latter value in-

	real-world		synthetic	
total	105	(100.0 %)	1'620	(100.0 %)
BASE	6	(5.7 %)	333	(20.6 %)
SHIFT	80	(76.2 %)	735	(45.4 %)
INS	53	(50.5 %)	713	(44.0 %)
MIN	81	(77.1 %)	810	(50.0 %)
HEUR	26	(24.8 %)	79	(4.9 %)

Table 2.7: Numbers of provably optimal schedules ($\ell = 0$) for the different algorithms.

creased when restricting the analysis to classification problems of higher capacity, e.g. to 70 % for problems with $C \geq 100$.

In Figure 2.6, the relative extension is plotted against the length $\tilde{h}$ of the corresponding intermediate schedule for the application of SHIFT to the synthetic problem instances. For SHIFT, which presents the best approximation algorithm, only 2.4 % of the final schedules have a relative extension of $\ell > 0.5$. They were all generated from a relatively short intermediate schedule of length $\tilde{h} \leq 20$. On the other hand, all the 735 schedules with $\ell = 0$ are rather short with $h \leq 14$ too. Besides, intermediate schedules of length $\tilde{h} > 40$ were produced only for the extreme classification problems with $(n, C) = (800, 10)$.

Intermediate Schedule Lengths

With the only exception of one synthetic instance, for all real-world and synthetic problems for which length- or weight-optimal IP solutions were found, the intermediate schedule of the approximation had the same length as the corresponding IP schedule. Thus, the relative excess matches the relative extension for these problems. Since the IP solver found weight-optimal schedules for all real-world problems, the numbers of zero-extension real-world schedules in Table 2.7 are *exact* numbers of optimal schedules found by the different algorithms. Note that this only holds for the real-world problems, the numbers in Table 2.7 for the synthetic instances only present lower bounds. As mentioned in Sect. 2.4.1, an optimal schedule satisfying (2.2) is longer than the optimal one for (2.4) in general, so the above result occurred rather unexpectedly.

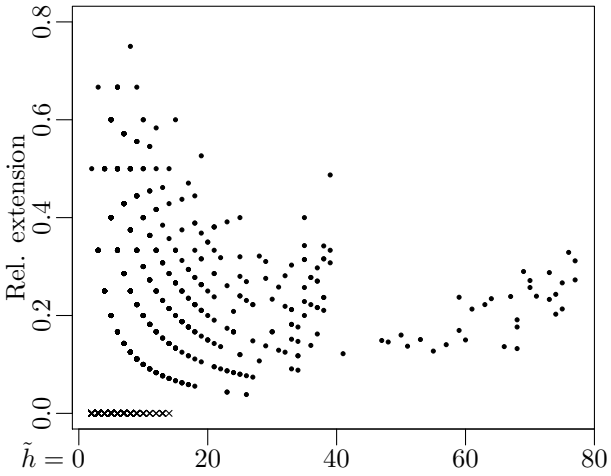


Figure 2.6: Relative extension of synthetic schedules produced by SHIFT in relation to the intermediate length $\tilde{h}$. $\times$: $\ell = 0$, $\bullet$: $\ell > 0$. The plot looks similar for the other variants of the approximation.

A possible explanation for this outcome is the following. The dynamic programming algorithm of Lemma 7 for deriving an intermediate schedule assigns the same bitstring to consecutive cars of the same train unless they form a break. (This fact is only accounted for by proving the algorithm's optimality w.r.t. (2.4) as mentioned in Section 2.4.1.) Hence, if an intermediate schedule violates (2.2) but not (2.4) for its length $\tilde{h}$, it will usually have rather long chains. The IP solver, however, may very well assign different bitstrings to two consecutive cars that are contained in the same chain. The required number of distinct bitstrings in the IP schedule is thus increased compared to the intermediate schedule, which assigns between $2^{\tilde{h}-1} + 1$ and $2^{\tilde{h}}$ different bitstrings to the cars. Still, the length of the optimal IP schedule will not exceed that of the intermediate schedule unless the required number of bitstrings in the optimal IP schedule exceeds $2^{\tilde{h}}$. All in all, the integer programming approach makes up for the stronger capacity constraints (2.2) by a higher flexibility to assign the available $2^{\tilde{h}}$ bitstrings of length $\tilde{h}$.

Running Times

For the 105 real-world problems, the IP solver computed all schedules in 103 seconds in total. This time covers the 105 successful iterations and proofs of weight-optimality but not the 392 attempts proved infeasible. The approximation algorithms and HEUR needed less than one second each for the complete set of 105 real-world problems.

For the whole set of 648 synthetic problems, the IP approach required more than 98 hours in total and did not find any feasible solution for 403 problems (confer Section 2.5.1). The three approximation algorithms BASE, SHIFT, and MIN as well as HEUR each needed less than one minute to compute the complete set of 1'620 synthetic problem schedules.

2.6 Summary

For the $\mathcal{NP}$-hard train classification problem with limited track capacities, an exact integer programming model has been introduced that applies the binary encoding of classification schedules to implement the feasibility and capacity requirements. Moreover, the approximation algorithm BASE with an approximation factor of two has been derived, including the two heuristicly improvements SHIFT and INS. The four approaches were compared to each other in a thorough experimental study, which additionally included the simple heuristic algorithm HEUR for comparison. It turned out that the approximations performed much better in practice than their approximation factors suggested, and even the basic algorithm BASE quickly yielded a feasible schedule when the IP approach needed a big number of iterations. However, applying such a rather involved algorithm is not justified since the much simpler heuristic HEUR rarely yielded longer schedules. Both BASE and HEUR are clearly outperformed by the heuristic improvements. With an average schedule length of only 5.4 % above optimal, SHIFT gets close to the real-world schedules of the slower IP approach, so very satisfactory practical schedules were obtained extremely quickly. Moreover, the inherent optimality condition of the approximation applied to 76.2 % of the real-world problems for SHIFT, so the approximation algorithms present a fast alternative for de-

riving capacity-restricted classification schedules with a highly competitive solution quality.

Chapter Notes The basic integer programming model of Section 2.3 was originally published in [MMN09], the basic approximation algorithm of Section 2.4.1 in [JMMN11] including Figure 2.1. The synthetic instances of Section 2.5 were derived in [MN09], which also contains the setup for computing IP schedules presented in Section 2.5.1. All other results from Sections 2.4.2, 2.4.3, and 2.5, including the figures, were originally presented in [HM10], which is a joint work with Alain Hauser, to which both authors contributed equal amounts.

Future Directions An exact algorithm might improve over the approximation by greater freedom in assigning bitstrings to cars and may still yield acceptable running times for small real-world instances.

Based on the same idea, improvements might also be achieved by a heuristic that results from modifying the first stage of the approximation algorithm: by splitting up very long chains, eliminating some violations of (2.2) may be achieved at the expense of an increased total weight.

The integer programming of Section 2.3 was mainly used to assess the approximation algorithms of Section 2.4, but the IP approach may also be improved by the approximation: for an intermediate schedule $\tilde{B}$ of length $\tilde{h}$, the weight $w(\tilde{B})$ of $\tilde{B}$ is a sophisticated lower bound for the IP objective of a solution attempt corresponding to $\tilde{h}$. As reported in Section 2.5.2, the solutions found by the IP solver usually had a length of $\tilde{h}$, so this lower bound may significantly accelerate finding weight-optimal IP solutions.

Chapter 3

Train Classification with Further Restrictions

In this chapter various constraints to the train classification problem are considered as announced in Section 1.2.3, adding to the restriction of limited track capacities dealt with in the previous chapter. Some of these constraints where mentioned in Section 1.1.4. A limited number of sorting tracks is dealt with in Section 3.2, which mainly provides an algorithm for computing an optimal schedule for this constraint. The operational restrictions of parallel sorting procedures, a varying number of output tracks, and train departures are considered individually in Section 3.3 and are integrated into an integer programming in Section 3.4. This extended integer programming approach is finally applied to a practical traffic instance in Section 3.5.

3.1 Overview and Related Work

An example for how to deal with a restricted number of classification tracks is given by Krell [Kre62]. This partially anticipates the generalized method introduced in Section 3.2, but does neither consider the order of inbound cars nor contain any optimality considerations. For the order specification of outbound trains defined in Section 1.2.2, the algorithm of Section 3.2 for com-

puting an optimal schedule is independently derived in [HZ07]. The authors further provide a closed-form expression for the number of valid bitstrings under this constraint.

For triangular and geometric sorting without considering the order of inbound cars, [Kre62] also contains upper bounds on the numbers of groups of the outbound trains that can be handled depending on the number of sorting tracks as mentioned in Section 1.2.6. Other publications mention that the different multistage methods have certain requirements on the number of classification tracks [Pen59], some also give method-specific numbers [Sid72, DDH83]. However, none does consider the order of inbound cars or provides solutions if the track requirements cannot be met.

A type of classification yard called "directional yard" is mentioned in [Pet77a]. This variant basically consists of two double-ended yards (see Figure 1.2(a)) which are physically arranged to serve opposite directions of a main line. This yields two mostly independent sorting procedures similarly to the situation considered in Section 3.3.1, but the main problem of assigning each outbound train to either procedure does not arise in [Pet77a].

For a related problem of allocating output tracks for unit trains in a classification yard, various practical constraints are considered in [BFMM11]. In particular, the output tracks are used several times for different outbound trains, and the constraints arising in connection with this are incorporated into a mixed integer programming model. Reusing sorting tracks as output tracks after they have been pulled in multistage sorting is mentioned in [Kre62] as a possible approach to reduce the number of required classification tracks. This policy yields the situation of an increasing number of output tracks discussed in more detail in Section 3.3.

Because outbound trains with the same numbers of chains are ready after the same number of sorting steps, simultaneous sorting in all its variants is described in [Kre62] as particularly beneficial for departure times of outbound trains being concentrated in a smaller interval. This fact is also brought in in [End63] as an argument for adhering to sorting by train for the example of the former classification yard of Coesfeld, Germany, with its departure times being distributed over the whole day. However, it is also remarked in [Kre63] that the connection

of departure times and applicability of simultaneous sorting is mostly overrated, particularly as the numbers of groups usually vary from train to train. Departure times of outbound trains are also considered in [BFMM11].

The applicability of the combined IP model of Section 3.4 is verified in the computer simulation of Section 3.5. Computer simulations present a useful tool for evaluating and refining classification methods before applying them in practice. Several such simulations have been performed in the recent past in order to verify planned modifications of yards or changes in operation for yards in Germany [EKMZ04], Slovakia [Már05], and Switzerland [ZL06].

3.2 Limited Number of Sorting Tracks

This section deals with the constraint of a limited number of sorting tracks. The precise setting is discussed in Section 3.2.1. An algorithm for deriving feasible schedules is given in Section 3.2.2, which is then proved to be optimal in Section 3.2.3.

3.2.1 Setting, Example, Notation

For the moment, this chapter returns to the setting with classification tracks of unrestricted capacities as announced in Section 1.2.2. (Track capacities are again considered from Section 3.4 on, where all the restrictions individually dealt with in Chapters 2 and 3 are considered in combination.) Moreover, the order of inbound trains is fixed, there are multiple outbound trains, and cars may have non-distinct types. The main constraint considered in this section is presented by a restricted number of sorting tracks. Recall from Section 1.2.2 that this number is denoted by W.

The length h^* of a schedule derived by Algorithm 1 is optimal for the setting without any constraints and thus presents a lower bound for any more general setting as mentioned in Section 1.2.5. As $h^* > W$ may hold, it immediately follows that sorting tracks are pulled more than once in general in this section. A corresponding classification schedule can still be represented by the assignment from cars to bitstrings introduced in Section 1.2.3. However, different from the situation

in Section 1.2.4, not every assignment from cars to bitstrings represents a schedule. This is illustrated in the following. In the example classification procedure of Figure 1.3, the schedule shown in Figure 1.3(l) has a length of $h = 4$, i.e., the assigned bitstrings have a length of $h = 4$. With regard to the more general setting of this section, this bitstring assignment clearly presents a feasible schedule for every number of sorting tracks $W \geq h = 4$. If the number of tracks is restricted, however, some instructions may not make much sense: assume, say, $W = 2$ and the two tracks are pulled out in alternating order. Then, car 5, for example, is initially to be sent to the track pulled in the third step ($b_2^5 = 1$) but not to the one pulled in the first step ($b_0^5 = 0$). However, it is the same physical track that is pulled out in the first and third step, which yields a contradiction. Put differently, the bitstring gives the instruction to neither send the car to the first pulled track nor to send it to the second at its initial roll-in, and the output track is not to be chosen either, so the instruction is not defined. Therefore, in general, not every bitstring corresponds to a valid instruction of how to handle a car in the classification process for a limited number of sorting tracks if $W < h$. How the bitstring representation can still be applied to derive feasible schedules for restricted numbers of sorting tracks is shown in the following section.

Since the sorting tracks are now pulled more than once in general, the order in which they are pulled must be specified. The W physical tracks can be indexed from 0 to $W - 1$ in an arbitrary order to uniquely identify each track. Then, for a classification procedure of h steps, the order in which the tracks are pulled out in the h successive sorting steps is given by a function $\theta\colon \{0, \ldots, h{-}1\} \to \{0, \ldots, W{-}1\}$ mapping each sorting step to a unique physical track. The sequence of indices $\theta(0), \ldots, \theta(h{-}1)$ will be called the *track sequence* of the classification procedure.

3.2.2 Deriving Feasible Schedules

If at least one sorting track is available, i.e., if $W > 0$, then every problem instance can be solved by the following brute force method. Let $y_1^1, \ldots, y_{k_1}^1, y_1^2, \ldots, y_{k_2}^2, \ \ldots\ , y_1^m, \ldots, y_{k_m}^m, y_1^{m+1}$ denote a chain decomposition of a given instance. At the initial roll-in every car of every chain $[\tau_{y_1^j}, \tau_{y_2^j}]$, $j = 1, \ldots, m$, i.e. all

the first chains of all the outbound trains, is sent to the output track of its respective train. Every other car is sent to the only sorting track, and this track is pulled out in every step of the succeeding sorting process. After the ith such sorting step, every car of every chain $[\tau_{y^j_{i+1}}, \tau_{y^j_{i+2}}]$, $j = 1, \ldots, m$, is sent to its output track, while all other car are sent back to the sorting track. By the definition of a chain decomposition, this approach represents a feasible classification schedule of length $h = k - 1$, where k denotes the length of the chain decomposition.

By the above considerations, the optimal length h^* of a feasible schedule for a limited number of sorting tracks ranges between $h^* = \lceil \log_2 k^* \rceil$ by Theorem 2 and $h^* = k^* - 1$, with k^* denoting the length of the greedy chain decomposition. In the following a more sophisticated algorithm for deriving feasible schedules is developed, which receives the number of sorting tracks W as a parameter with its input. For the extremal cases of $W = 1$ or a large value of W, the output of this algorithm corresponds to the simple procedure of the previous paragraph or the schedule computed by Algorithm 1, respectively. It turns out that this generalized algorithm yields optimal schedules as shown in Section 3.2.3.

As mentioned before, each sorting track is pulled out more than once in general. The order θ in which the tracks are pulled can be described as a "round robin" order: a track is only pulled for its ith time if every other track has been pulled at least $i-1$ times; also, the order in which the tracks are pulled in the ith round of pull-outs is the same as for the $(i-1)$th round. More precisely, for a sorting process of h steps, the order in which the W sorting tracks are pulled is called *round robin* if, for every sorting track $k \in \{0, \ldots, W-1\}$, there is an integer $0 \leq r_k < W$ such that for every step $i = 0 \ldots, h-1$ the equation $\theta(i) = k$ holds if and only if $i \equiv r_k \pmod{W}$. W.l.o.g., r_k is assumed to be the track's index k. That is to say, for every sorting track $k \in \{0, \ldots, W-1\}$ and every sorting step $i = 0 \ldots, h-1$, $\theta(i) = k$ if and only if $i \equiv k \pmod{W}$.

The following lemma states how many valid bitstrings of a given length there are for a limited number of sorting tracks and a round robin order. Its proof implies a recursive procedure for constructing these bitstrings, which is used for deriving a feasible schedule further below.

Lemma 8 *Given nonnegative integers W and h, the number of bitstrings of length h that represent valid handling instructions of cars for a classification schedule pulling W available sorting tracks in a round robin order is given by the following recurrence formula:*

$$R_W(h) = \begin{cases} 2^h & \text{if } h \leq W \\ 1 + \sum_{i=1}^{W} R_W(h-i) & \text{if } h > W \end{cases} \tag{3.1}$$

Proof Let $b = b_{h-1} \dots b_0$ be a bitstring of length h assigned to any car. Whenever this car is rolled in, be it the initial roll-in or a roll-in after any pull-out, it must be sent to one of the W available sorting tracks or to the output track of its outbound train. The initial roll-in will be regarded as succeeding the 0th pull-out step in the following considerations, which can be treated as $b_{-1} = 1$. After the ith pull-out step of any order θ, each car on the lead track must be sent to one of the W sorting tracks. The track indexed k, $k \in \{0, \dots W-1\}$, is pulled next in step $\min\{j = i, \dots, h \mid \theta(j) = k\}$. For a round robin order, this equals $\min\{j = i, \dots, h \mid j \bmod W = k\}$. By the equality $\{i \bmod W, (i+1) \bmod W, \dots, (i+W-1) \bmod W\} = \{0, \dots, W-1\}$, sending a car to one of the W sorting tracks in the round robin order means sending to a track pulled in one of the next W steps. Hence, if a car is about to be rolled in after $b_{i-1} = 1$, its bitstring b must satisfy $b_{i-1+k} = 1$ for some $k \in \{1, \dots, W\}$ unless it is sent to the output track. Sending the car to the output track, however, corresponds to $b_{i-1+k} = 0$ holding for all $k = 1, \dots, h - i$.

In other words, bitstring b is only valid if there is no sequence of W or more consecutive zero-bits that are not leading. For the special case $h \leq W$, a non-leading sequence of W consecutive zero-bits cannot occur, so $R_W(h) = 2^h$, which proves the base case of Formula 3.1.

The case $h > W$ of longer bitstrings is depicted in Figure 3.1. First of all, if the car is not initially sent to its output track, there are W choices of rolling it in to a sorting track, which means that b must satisfy $b_i = 1$ for some $i = 0, \dots, W - 1$. Let now i be the smallest such index in b, corresponding to one of the W rows shown in Figure 3.1. For the roll-in succeeding this pull-out step $b_i = 1$, there are again W choices of

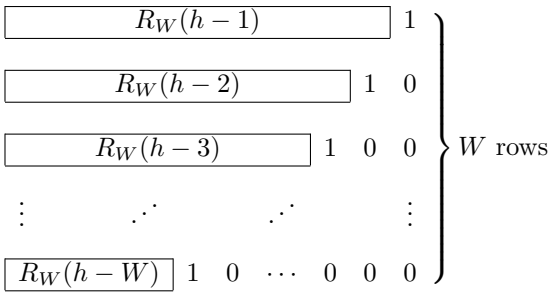


Figure 3.1: Recursive construction of valid bitstrings of length h from shorter valid bitstrings for pulling W sorting tracks in round robin order.

a sorting track to which to send the car assigned to b. The number of bitstrings b with a tail $b_i \dots b_0 = 10 \dots 0$ is thus given by $R_W(h-i-1)$ for this value of i. Summing over all the W trailing such segments, corresponding to the rows in Figure 3.1, yields $R_W(h) = \sum_{i=0}^{W-1} R_W(h-i-1)$, which equals $\sum_{i=1}^{W} R_W(h-i)$. In addition, there is the possibility to send the car to the output track of its outbound train. This means that b consists of zero-bits only, which yields one more bitstring in this step of the recursion. This totals to a number of bitstrings $R_W(h) = 1 + \sum_{i=1}^{W} R_W(h-i)$, which finishes the general case of Formula 3.1. □

Note that exactly the same values $R_W(h)$ are calculated if the recurrence formula in (3.1) for $h > W$ is extended to cover all the cases $h \geq 1$. The base rule $R_W(h) = 2^h$ for $h \leq W$ then applies only for $h = 0$ anymore and simplifies to $R_W(h) = 1$ for $h = 0$. However, having an explicit number $R_W(h) = 2^h$ of bitstrings for all the cases $h = 0, \dots, W$ will come in handy further below.

The value of $R_W(h)$ can be calculated by a dynamic programming approach. The table consists of a single row of $h+1$ entries $R[0], \dots, R[h]$ and is initialized with $R[i] = 2^i$ for $i = 0, \dots, W$. The remaining entries are successively calculated according to the second rule of Formula 3.1, each in time $\mathcal{O}(W)$. Hence, the total time is $\mathcal{O}(W + (h-W)W) = \mathcal{O}(hW)$.

Accordingly, given W and a minimum number of required bitstrings s, the minimum value of h for which $R_W(h) \geq s$ can be computed using the same approach. For this value of h, the corresponding set S of all valid bitstrings of length h can then be derived with the procedure implied by the proof of Lemma 8: if $h \leq W$, then S is given by the set of all bitstrings of length h. Otherwise, apply the construction depicted in Figure 3.1, but in a bottom up fashion: first, successively derive all bitstrings of length i for $i = 0, \ldots, W$; then, successively apply the construction depicted in Figure 3.1 for the length values from $W + 1$ to h, respectively, using the required shorter bitstrings derived in earlier steps. The total number of bitstrings generated during this process is at most:

$$|S| + \sum_{i=0}^{W} 2^i = R_W(h) + 2^{W+1} = R_W(h) + 2R_W(W) < 3R_W(h)$$

Constructing a single such bitstring takes $\mathcal{O}(h)$ corresponding to its length. Hence, the total computation time is $\mathcal{O}(hR_W(h))$, which is linear in the size of the output.

Finally, a procedure for deriving a feasible schedule for a limited number of tracks is obtained. First, a greedy chain decomposition $y_1^1, \ldots, y_{k_1}^1, y_1^2, \ldots, y_{k_2}^2, \ \ldots \ , y_1^m, \ldots, y_{k_m}^m, y_1^{m+1}$ is computed using Algorithm 1. Let k^* denote the length of the chain decomposition. Second, the minimum value of h for which $R_W(h) \geq k^*$ is calculated as well as the set S for a minimum number of required bitstrings k^*. Finally, instead of assigning the bitstring $[i-1]_2$ to all cars of chain $[\tau_{y_i^k}, \tau_{y_{i+1}^k}]$ for every train $k = 1, \ldots, m$ as done in line 12a of Algorithm 1, the ith smallest bitstring of S (w.r.t. order relation (1.1)) is assigned. The size of S clearly suffices for doing this, and the sequence of bitstrings assigned to successive chains is increasing w.r.t. (1.1). Therefore, the assignment yields a feasible schedule of length h for a limited number W of sorting tracks. In fact, the assignment yields a feasible schedule of minimum length, which is shown in the following section.

3.2.3 Optimal Schedules

In the construction of the previous section, bitstrings of a certain length h from a set S are assigned to the chains of the

greedy chain decomposition. The greedy chain decomposition has minimum length k^*, and S is constructed such that every set S' of round robin bitstrings of a length less than h has a size of less than k^*. Therefore, the corresponding schedule is optimal w.r.t. a round robin order of pulling the sorting tracks.

In Lemma 8 a maximum set S of bitstrings was constructed that represent valid classification instructions for single cars in a procedure with round robin order. This construction can be generalized to any order θ of pulling the sorting tracks as shown in the following theorem. It will turn out that no order yields a set of bitstrings that is larger than that of the round robin order, which shows that using round robin leads to optimal schedules.

Theorem 7 *Given a train classification problem, there is a feasible schedule of minimum length w.r.t. a limited number W of sorting tracks in which the tracks are pulled in a round robin order.*

Proof Let $\theta : \{0, \ldots, h-1\} \to \{0, \ldots, W-1\}$ be any order of pulling the W sorting tracks in a classification procedure of h steps. In order to derive a maximum set S of valid round robin bitstrings, the reasoning for round robin orders in the proof of Lemma 3.1 can be generalized. Consider the subbitstring $b_{i-1} \ldots b_0$ with $b_{i-1} = 1$ of any valid $b = b_{h-1} \ldots b_0$ to be derived. In the roll-in following $b_{i-1} = 1$, any car assigned to b must be sent to any track k, $k \in \{0, \ldots W-1\}$, for which $\theta(x) = k$ for some step $x \geq i$, i.e. to any track that is still to be pulled in some later step. Regarding the W possible choices for k yields a picture similar to that of Figure 3.1: there are up to W rows corresponding to at most W choices; since the kth track is pulled next in step $j := \min\{j' = i, \ldots, h-1 \mid \theta(j') = k\}$, choosing k corresponds to putting $b_j b_{j-1} \ldots b_0 = 10 \ldots 0$ for b in the row corresponding to the kth track. There are $h - 1 - j$ trailing bits left in this row to complete b, for which there are $R^\theta_W(h-1-j)$ possibilities if $R^\theta_W(x)$ denotes the number of valid bitstrings of length x for the order $\theta|_{\{h-x,\ldots,h-1\}}$. The sum of the R^θ_W-values over the rows gives the number of valid bitstrings of length $h - i$, and considering $i = 0$ gives the total number of valid bitstrings b of length h. There are at most W rows in every recursive step, corresponding to the size of the subset of tracks that are still to be pulled in later steps. The round robin

order yields the maximum number of rows in every step, which is given by $\min(W, h-i)$ in step i. Hence, for any length h, the round robin order yields a maximum set S of valid bitstrings of length h. Therefore, if there is a feasible schedule of length h, there also is a feasible schedule of length h for which the tracks are pulled in round robin order. □

3.3 Further Restrictions in Practice

The number of sorting tracks and their capacities are assumed unrestricted again in this section, so every involved sorting track is pulled exactly once. The order of inbound trains is fixed, and there are multiple outbound trains. Considering distinct cars or cars with types does not make any difference in this section. In addition to the infrastructural constraints highlighted in Section 1.1.4, there may apply various yard-specific operational constraints. Some of these are individually treated in this section before integrating them in Section 3.4 below.

3.3.1 Parallel Sorting Procedures

There are classification yards that feature two parallel lead tracks. An example is the mentioned yard Lausanne Triage considered in the case study of Section 3.5. Every classification track is connected to both lead tracks. Until the first pull-out of the multistage method is performed, i.e. during primary sorting, the two lead tracks are used alternately for rolling in cars. While a train is pushed over the hump from either lead track, another train is brought from the receiving yard to the other lead track and prepared for the following roll-in. In this way, the sequence of rolling in all the inbound trains is accelerated simply by reducing the idle time of the hump.

Moreover, the parallel lead tracks may be used to apply secondary sorting of two separate multistage processes in parallel.

Either process is assigned a fixed set of sorting tracks such that no path from one lead track to any of its classification tracks crosses any path from the other lead track to its assigned tracks. The set of outbound trains is then partitioned by assigning each train to either system, and the cars of the inbound trains are

rolled in to the set of tracks of the process to which they so belong. Then, two shunting engines are used to independently pull out and roll in the cars of the two partitions in parallel.

The total number of pull-out steps then equals $\hat{h} + \check{h}$, where $\hat{h}$ denotes the number of steps applied in the one system and $\check{h}$ the number of steps in the other. As has been shown in the example of Figure 1.3, the length of a schedule (in one system) is determined by the length of the longest subschedule of its outbound trains. If the same procedures are simultaneously applied in one system instead, the number of steps is therefore given by $\max(\hat{h}, \check{h}) \leq \hat{h} + \check{h}$. Hence, the number of pull-out steps and thus, as every track is pulled only once, also the number of required sorting tracks, is always minimized by assigning all outbound trains to a common partition. Thus, parallelizing yields no actual improvement here.

The turnaround time of a classification process with two partitions, meaning the time when the last outbound train has been finished, is given by $\max(\hat{h}, \check{h})$, i.e. the number of steps in the partition with the longest subschedule. Therefore, if the turnaround time is considered, all partitions are equivalent. In the setting with capacities, this fact can be used to trade the number of sorting tracks for a lower capacity requirement. For example, the sorting procedure of Figure 1.3 requires sorting tracks of capacity $C \geq 9$ as the track pulled in Figure 1.3(g) contains nine cars at this time. When the respective partial schedules of the yellow and grey outbound train are performed in separate systems as explained above, the required capacity reduces to $C = 6$, while the number of sorting tracks increases from four to seven. The turnaround time remains at $\max(4, 3) = 4$.

Similarly, instead of accepting a high number of steps (for a single shunting engine) to obtain a feasible schedule that is feasible w.r.t. a low capacity constraint (2.2), two shorter, parallel procedures may be applied without changing the capacity requirement and the *total* number of sorting tracks. This approach is taken in the case study of Section 3.5.

3.3.2 Number of Output Tracks

It has been explained in Section 1.1.3 how the multistage method and other sorting activities are performed in a shared infrastructure. In particular, the cars involved in multistage sorting

are initially rolled in to a predefined set of classification tracks, while other sorting activities are performed in the rest of the yard. Primary sorting is stopped at some point and the hump is exclusively used for the actual multistage sorting process, while outbound trains not involved in the multistage method successively leave the classification bowl and free their respective classification tracks. The ongoing multistage sorting process may then include these released tracks for forming outbound trains, so the number of available output tracks may increase over time. When a classification track has been cleared, cars from the roll-in operation starting after this point of time may be sent to it. Time can thus be discretized according to pull-out steps by defining N_t as the number of available output tracks after the tth pull-out step, $t = 0, \ldots, h$. Put differently, a total of not more than N_t outbound trains may have started to be formed in the tth sorting step. An outbound train is said to be started to be formed when its first car arrives at the output track.

If this has to be dealt with as a sole restriction, consider the optimal schedule B^* calculated by Algorithm 1. After the h^*th step, all the m outbound trains are finished, in particular they have all been started to be formed. If $m > N_{h^*}$, the number of output tracks does not suffice. B^* must thus be extended by adding a 1-column as the (h^*+1)th step at the left of B^*. In this way, no train is started before the (h^*+1)th step and all trains are finished after this sorting step. This is iterated until $m \leq N_h$ for some $h \geq h^*$. Note that this approach does not minimize the number of roll-ins.

3.3.3 Train Departure Times

The outbound trains may have departure times. Time has been discretized in the previous section by letting one time unit correspond to one sorting step. With this, the departure time of an outbound train yields the latest sorting step in which this train must be finished, i.e. an upper bound on the length of the partial schedule of this train. If this presents a sole restriction to the classification schedule, an optimal schedule B^* computed by Algorithm 1 must be checked to meet all the train departures. If it does not, there is no feasible schedule satisfying the restriction since B^* minimizes the length of every partial schedule by the assignment of line 12a.

Train departure times become particularly interesting if they occur in combination with time-dependent numbers of available output tracks, which were introduced in the previous section, as they present diametrical restrictions: for each train, the train departure times define a latest step for finishing, whereas the numbers of available output tracks yield a set of earliest steps of starting train formation to be distributed to the outbound trains. This combination is considered in integrated integer programming approach of the following section, which also incorporates the parallel sorting procedures from above.

3.4 Extended Integer Programming Approach

The practical restrictions individually treated in the previous section are combined in the following. In order to integrate them into the general problem with restricted track capacities, the basic integer programming model of Section 2.3 is successively refined in this section. The resulting extended IP model will then be applied in a case study in the subsequent section.

3.4.1 Parallel Sorting Procedures

First, the IP model is extended by incorporating two simultaneous classification procedures on two parallel lead tracks. Recall that these procedures work as two independent systems with one shunting engine in either system and each sorting track is used by only one procedure. In particular, every outbound train is assigned to exactly one of the systems and remains in that system from its first roll-in until its outbound train is formed. The two systems will be called *north partition* and *south partition*, following the situation of Lausanne Triage, in which one partition is geographically located north of the other.

The assignment of trains to partitions is part of the optimization process. The binary variables s_i, $i = 1, \ldots, m$, indicate the trains' respective system memberships by $s_i = 1$ if the ith outbound train is a member of the north partition and $s_i = 0$ otherwise. Moreover, the binary schedule variables b_i^τ are doubled into two sets, with $\hat{b}_i^\tau$ for the schedule corresponding to

the north and $\check{b}_i^\tau$ to that of the south partition. In the resulting model h sorting steps are performed in each partition. Let further denote $t(\tau)$ the outbound train of car $\tau \in \{1,\dots,n\}$.

With this, the integer programming model can be extended to two independent sorting procedures:

$$\min \sum_{i=0}^{h-1}\sum_{\tau=1}^{n}\left(\hat{b}_i^\tau + \check{b}_i^\tau\right)$$

$$\text{s.t.} \sum_{i=0}^{h-1} 2^i \hat{b}_i^\tau \geq \operatorname{rev}(\tau,\tau-1) - \left(1-s_{t(\tau)}\right) + \sum_{i=0}^{h-1} 2^i \hat{b}_i^{\tau-1},\ \tau \notin F \tag{3.2}$$

$$\sum_{i=0}^{h-1} 2^i \check{b}_i^\tau \geq \operatorname{rev}(\tau,\tau-1) - s_{t(\tau)} + \sum_{i=0}^{h-1} 2^i \check{b}_i^{\tau-1},\ \tau \notin F \tag{3.3}$$

$$\sum_{\tau=1}^{n} \hat{b}_i^\tau \leq C,\ i \in \{0,\dots,h-1\} \tag{3.4}$$

$$\sum_{\tau=1}^{n} \check{b}_i^\tau \leq C,\ i \in \{0,\dots,h-1\} \tag{3.5}$$

$$\hat{b}_i^\tau, \check{b}_i^\tau \in \{0,1\}, \quad i \in \{0,\dots,h-1\},\ \tau \in \{1,\dots,n\} \tag{3.6}$$

Note this approach assigns *two* bitstrings $\hat{b}^\tau$ and $\check{b}^\tau$ to car τ, one for either partition. Consider any pair of consecutive cars τ and $\tau-1$ that both belong to the kth outbound train and appear in reversed order in the inbound sequence of cars. If k is assigned to the north partition, i.e. $s_k = 1$, then $1 - s_{t(\tau)} = 1 - s_k = 0$ and constraints (3.2) simplify to the ordering constraints (2.1) of the original IP of Section 2.3. In this case, the values of $\check{b}^\tau$ and $\check{b}^{\tau-1}$ have no meaning at all, and constraints (3.3) are satisfied if both $\check{b}^\tau = 0$ and $\check{b}^{\tau-1} = 0$ independently of the value of $\operatorname{rev}(\tau,\tau-1)$. Hence, the objective function will cause $\check{b}^\tau = 0$ and $\check{b}^{\tau-1} = 0$ in any optimal solution, and the objective value actually equals the weight of the schedule. A similar argument applies for $s_k = 0$, when the ordering constraints (3.3) apply instead of (3.2). Note that constraints (3.2) and (3.3)—as well as constraint (2.1) of the original IP—ensure the solution to comply with the requirements of Theorem 1 for feasibility even for general instances with cars of non-distinct types. The original capacity constraints (2.2) for a single sorting procedure straightforwardly generalize to (3.4) and (3.5) as each sorting track contains cars of only one partition.

3.4.2 Available Classification Tracks

For the availability of classification tracks, the two phases of primary and secondary sorting have to be distinguished for multistage sorting. As illustrated in Section 1.1.3, the cars involved in multistage sorting are collected on a set of W reserved classification tracks before the first pull-out step is performed. In order to model the two parallel procedures, let $\hat{W}$ and $\check{W}$ denote the numbers of tracks reserved in the north and south system, respectively, with $\hat{W} + \check{W} = W$.

The reserved sets of tracks can then be integrated into the integer programming model by the following constraints:

$$\sum_{i=0}^{\hat{W}} \hat{b}_i^\tau \geq s_{t(\tau)}, \qquad \tau \in \{1, \ldots, n\} \tag{3.7}$$

$$\sum_{i=0}^{\hat{W}} \check{b}_i^\tau \geq (1 - s_{t(\tau)}), \qquad \tau \in \{1, \ldots, n\} \tag{3.8}$$

Note that for the special case of $h = \hat{W} = \check{W}$ this simply means disallowing the all-zero bitstring for every car, so cars may not be sent to their output tracks in primary sorting. This also holds for the sample classification instance of the case study of Section 3.5.

In secondary sorting, these tracks are successively pulled out, and forming outbound trains may be started—to the extent to which output tracks are available. For time discretized according to pull-out steps, the number of available output tracks at time t, $t = 0, \ldots, h$, is denoted by N_t in Section 3.3.2 and assumed to be nondecreasing. To express the number of outbound trains the cars of which have been sent to the respective output track in the integer program, binary variables $\hat{u}_{k,t}$ and $\check{u}_{k,t}$ are introduced. For $k = 1, \ldots, m$ and $t = 0, \ldots, h$, variables $\hat{u}_{k,t}$ and $\check{u}_{k,t}$ indicate whether forming the kth outbound train has been started after the tth sorting step in the north and south partition, respectively.

The following constraints relate the number of started outbound trains for each step to the number of available output tracks:

$$\sum_{\tau \in F} \hat{u}_{t(\tau),t} \leq \hat{N}_t, \quad t \in \{0, \ldots, h\} \tag{3.9}$$

$$\sum_{\tau \in F} \check{u}_{t(\tau),t} \leq \check{N}_t, \quad t \in \{0, \ldots, h\} \tag{3.10}$$

$$\hat{u}_{t(\tau),t} \geq s_{t(\tau)} - \sum_{i=t}^{h-1} \hat{b}_i^\tau, \quad \tau \in F, \ t \in \{0, \ldots, h\} \tag{3.11}$$

$$\check{u}_{t(\tau),t} \geq 1 - s_{t(\tau)} - \sum_{i=t}^{h-1} \check{b}_i^\tau, \quad \tau \in F, \ t \in \{0, \ldots, h\} \tag{3.12}$$

After every step t, the number of started outbound trains must not exceed the available number $\hat{N}_t$ or $\check{N}_t$ of tracks, respectively, at this time. This is implemented by constraints (3.9) and (3.10). Constraints (3.11) and (3.12) ensure that each variable $\hat{u}_{k,t}$ and $\check{u}_{k,t}$ is actually set if forming the kth train has been started in the respective partition in the tth step. This is the case if and only if the train is assigned to the respective partition and its first car, i.e. the car $\tau \in F$ with $t(\tau) = k$, has been sent to the output track. This corresponds to leading zeroes $\hat{b}_{h-1}^\tau \ldots \hat{b}_t^\tau = 0 \ldots 0$ or $\check{b}_{h-1}^\tau \ldots \check{b}_t^\tau = 0 \ldots 0$, respectively, in the bitstring assigned to τ for its partition.

3.4.3 Train Departure Times

If outbound trains come with departure times as described in Section 3.3.3, each such continuous time value can be assigned to the discrete sorting step preceding it. In this way, a latest sorting step acc_k in which the kth outbound train may still receive cars is obtained for every $k = 1, \ldots, m$.

This finally yields the following last constraint for the extended integer programming model, which simply disallows any roll-ins of cars after the departure time of their respective outbound train:

$$\sum_{i=\mathrm{acc}_{t(\tau)}}^{h-1} \left(\hat{b}_i^\tau + \check{b}_i^\tau \right) = 0, \quad \tau \in \{1, \ldots, n\} \tag{3.13}$$

In the following section, this extended model is used to derive schedules for classification problems of the hump yard Lausanne Triage, to which all the constraints (3.2) to (3.13) must be

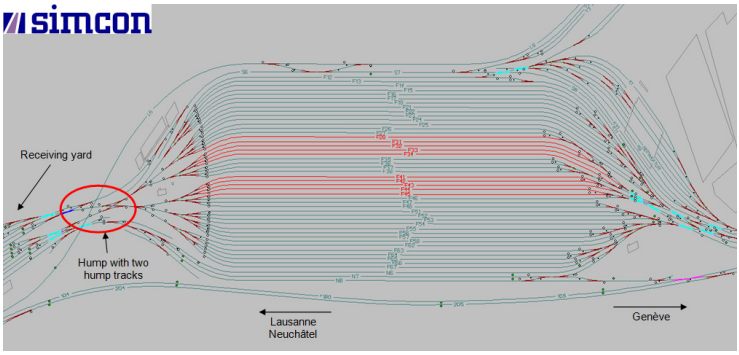


Figure 3.2: Classification bowl layout of Lausanne Triage with ten classification tracks (in red) reserved for multistage sorting.

applied. Details about all the instance-specific parameters are also given there, including the actual numbers of cars and trains, sorting and output tracks, capacities, and departure times.

3.5 Case Study: Lausanne Triage

In this section the IP model introduced just above is applied to the practical train classification instances described in Section 3.5.1. The precise setup for computing schedules is presented in Section 3.5.2 and the computed schedules in Section 3.5.3. One of these schedules is then further analyzed in a computer simulation as described in Section 3.5.4.

3.5.1 Classification Yard Lausanne Triage

The hump yard of Lausanne largely follows the standard layout described in Section 1.1.1. It features a receiving yard, a classification bowl of 38 classification tracks (depicted in Figure 3.2) with two parallel lead tracks as in Section 3.3.1, but no departure yard. The classification tracks of the yard have capacities corresponding to about $C = 40$.

Regarding the operation, there are ten classification tracks reserved as sorting tracks, which are colored red in Figure 3.2. All cars for the multistage method are initially collected on these

tracks, while other shunting is done on the remaining classification tracks. From some point in the early morning, the humps are exclusively used for multistage sorting. Then, the situation described in Section 3.4.2 applies, when more and more tracks become available for forming outbound trains.

Schedules are computed according to the approach explained in Section 3.5.2 for the five classification problem instances of Table 2.1, which were extracted from a week of traffic of the classification yard Lausanne Triage in Switzerland.

3.5.2 Schedule Computation

All IP computations were performed with ILOG Studio 3.7 featuring CPLEX 9.0 on an Intel Xeon CPU with 2.80 GHz and 2 GB main memory.

The computations were carried out similarly to the experiments described in Section 2.5: the schedule length h, which is the primary objective, presents a parameter also in the extended IP model, and the model's objective minimizes the weight of the schedule. Starting from the lengths of the respective schedules originally applied to the classification instances of Table 2.1, models with decreasing length values were successively passed to the solver until no feasible schedule was obtained. If no feasible schedules were found for some length values, attempts were made to derive feasible schedules for slightly increased capacities $C > 40$. When the actual usage of sorting tracks w.r.t. the maximum sum of lengths of cars for each track is analyzed for such a schedule, the track capacities may incidentally not be violated by a favorable distribution of the deviation of car lengths. The original schedules applied in 2005 all comprised five steps in each partition, which corresponds to $h = 5$ in the extended model of Section 3.4.1.

Furthermore, some of the outbound trains of the classification instances contain some destinations for which there is a very big number of cars. In the original schedule, these cars were not rolled in to the ten classification tracks for multistage sorting but directly to their respective destination tracks. (These cars are not contained in the numbers of Table 2.1.) This approach is suggested for cars of large groups in [Kre62] to avoid sending many cars over the hump for a second time, which presents a heuristic measure to reduce the classification effort. However,

for destinations located in the middle of an outbound train, some extra shunting must be done to assemble the outbound train as described in Section 1.1.3. Most of the large destinations in the given test instances are at the very front of their respective outbound trains, so the classification process is mostly not impaired by this practice. For the sake of an easier comparison, the same approach is taken here. In order to avoid interfering with the concurrent operation of shunting activities other than multistage sorting, the large destinations are sent to the same tracks as in the original schedules. This also requires a fixed assignment to the north or south partition for the affected outbound trains, which is done by forcing $s_k = 0$ or $s_k = 1$, respectively, for each such train k in the IP model: seven trains each are fixed in MON, WED, and THU, respectively, eight in TUE, and none in WKD.

Unfortunately, the values for $\hat{N}_t$ and $\check{N}_t$ for $t \in \{0, \ldots, h\}$ become only available by performing a simulation as done in Section 3.5.4. Such a simulation is performed in order to verify and further analyze a schedule derived with the extended integer programming model. However, this IP model itself contains $\hat{N}_t$ and $\check{N}_t$ as parameters. Hence, a first schedule is derived for putting $\hat{N}_t = \check{N}_t = \infty$, which is fed into the simulation tool. The simulation fails at some point, up to which the correct values of $\hat{N}_t$ and $\check{N}_t$ are available then. A further schedule is then obtained with the updated IP model, which can again be passed to the simulation tool. In each such iteration further values of $\hat{N}_t$ and $\check{N}_t$ for higher values of t become available until the simulation eventually finishes for some iteration. Since starting a single iteration of a simulation, this iterative procedure was carried out for only one of the instances, TUE, as reported in Section 3.5.3. This is the only instance for which a full simulation was performed.

3.5.3 Computed Schedules

For MON an improvement from five steps in both systems to a feasible schedule with five steps in the south and only four in the north system was obtained for a capacity value of $C = 54$. For this schedule, however, the track capacities of Lausanne Triage were exceeded for three tracks, so no improvement is achieved for MON. Similarly, for WKD a feasible schedule with only four steps

in each system was derived putting $C = 46$, but the capacity of a track was largely exceeded in this schedule.

A feasible schedule with four steps in each system was derived for WED that satisfied the capacity constraints for $C = 40$. For THU and WKD, feasible schedules were found with five steps in the north and four in the south system. Both satisfied their capacity constraints with $C = 41$, but this did not cause any overfull track when the actual filling of the tracks was analyzed. For TUE there is a feasible schedule with only four steps in each system for $C = 42$ which did not cause any overfull track. However, in the simulation it turned out that there are three output tracks missing after the fourth sorting step in this schedule.

Still, a feasible schedule satisfying the capacity constraints for $C = 40$ was achieved with five steps in the north and four in the south system that also respected the available numbers of output tracks at any time. Computing this schedule took 5.75 hours including the proof of optimality. This schedule is presented in further detail in Section 3.5.4.

3.5.4 Simulation

The feasible schedule with five steps in the north and four steps in the south system, which was derived for $C = 40$ as described at the end of the previous section, was further analyzed as described in this section. To this end, the simulation system "Villon" [AKK07, Vil] was applied. First of all, the schedule did not produce any conflicts when the simulation was run on it. This basically means, with regard to the technical implementation, that the schedule works in practice.

The total number of cars rolled in during the complete improved classification procedure amounts to 1'700, compared to 1'706 for the schedule applied originally, which is only a marginal saving. Nevertheless, as mentioned in Section 1.1.4, increasing the number h of steps in the multistage method over the optimum value generally allows decreasing the total number r of cars rolled in and vice versa. In the light of this correlation, the marginal reduction of r by six must be regarded as a great success since the reduced number of sorting steps does not have to be paid for with more roll-ins compared to the original schedule. This finding also underlines the suboptimality of the schedule originally applied.

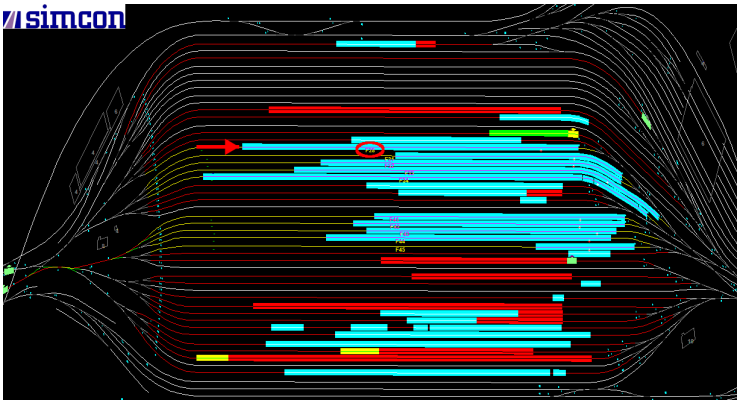


Figure 3.3: Situation of the cars on the classification tracks after the initial roll-in for the originally applied schedule. North is at the bottom of the picture.

If consecutive cars on the lead track are about to be rolled in to the same classification track, it is common practice not to decouple them. In accordance with a definition in [DDH83], a small set of such coupled cars pushed over the hump will be called a *cut*. If such a cut rolls over a switch, the switch will be worn, which is also caused by resetting the switch. The number of settings of switches for the improved schedule (including not only multistage sorting but all shunting) amounts to 789 compared to 914 for the old schedule, which is a considerable saving of 125 settings or 13.7 %. This significantly reduces the wear of the switches and saves maintenance, to which is further contributed by only 1'481 movements of cuts over switches. Compared to 1'691 for the original schedule, this is a saving of 210 cuts or 12.4 %.

The main improvement of the new schedule, however, consists in saving one full sorting step: in the original procedure the track labeled "F28" in Figure 3.3 contains the cars that were pulled out in the fifth sorting step of the south system. (Note that south is at the top of Figures 3.3 and 3.4.) The improved procedure only performs four steps in this system, and the track corresponding to the fifth pull-out remains empty after the initial roll-in, which can be seen in Figure 3.4. The track

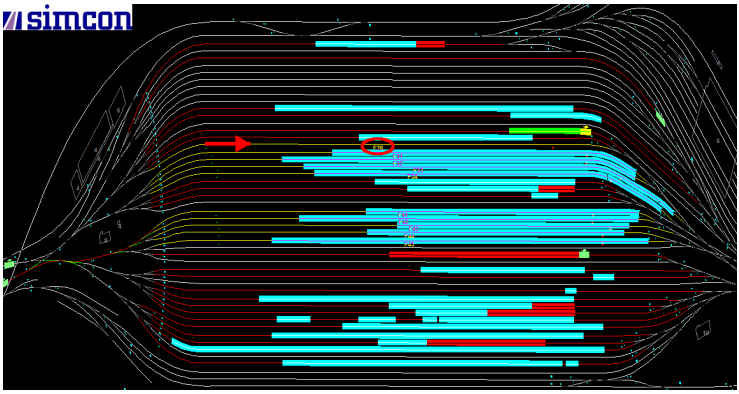


Figure 3.4: Situation of the cars on the classification tracks after the initial roll-in for the improved schedule computed in Section 3.5.3.

made available in this way can be used, for example, for multi-stage sorting in order to increase the upper limit of traffic with a higher attractiveness for this method through an increased potential traffic volume. The track may also be used for other shunting activities, such as building very long unit trains by collecting their cars from several classification tracks as illustrated in Section 1.1.3.

3.6 Summary

Various practical constraints for train classification introduced in Section 1.1.4 have been dealt with in this chapter. First of all, an optimal algorithm has been derived for computing shortest feasible schedules for the problem variant with a limited number of sorting tracks. This shows that a limited number of sorting tracks yields an algorithmically less difficult problem than the limited track capacities considered in Chapter 2. Moreover, it has been shown how to deal with parallel lead tracks, an increasing number of output tracks, and fixed departure times of outbound trains. Finally, the individual constraints have been integrated in the exact integer programming approach for restricted capacities of Section 2.3, which shows the high flexibility of this approach and thus applicability in practice.

As the main result, a feasible schedule has been successfully derived for the real-world traffic instance of the classification yard Lausanne Triage that outperforms the current schedule by one sorting step. This yields a more efficient sorting process, less engine movement, reduced wear of switches, and, most importantly, one more free classification track, raising potential for more traffic to be handled in the yard by either multistage sorting or any other shunting method applied.

Chapter Notes Deriving optimal schedules for a limited number of tracks as shown in Section 3.2 was originally published in [JMMN07, JMMN11], including Figure 3.1. As mentioned in Section 3.1, Hansmann and Zimmermann independently obtained the same algorithm in [HZ07]. Both publications consider the order specification of outbound trains defined in Section 1.2.2 with no track capacities and cover the case of types of cars.

The considerations of Section 3.3 have not been published before. The advanced IP model of Section 3.4 was developed in [MMN09], which also contains the case study of Section 3.5 including the figures in that section. The additional numbers of Section 3.5.3 were previously unpublished.

Future Directions For Lausanne Triage dropping the fixed assignment of some trains to partitions mentioned in Sect. 3.5.2 may yield an even better schedule with higher savings. It would furthermore be interesting to derive and simulate more schedules for further real-world data, particularly for larger classification yards than Lausanne Triage with higher volumes of traffic for multistage sorting. For such yards even higher improvements can be expected.

Beyond that, the practice of applying single-stage and multistage sorting strictly sequentially as described in Section 1.1.3 restricts the potential of optimization. This may be overcome by abandoning this separation in favor of an integrated method. This requires considering the arrival and departure times of trains, a first step to which direction has been made in this chapter by incorporating train departures. A full generalization of the train classification problem to time-dependent input may allow combining the strictly separated methods of single- and

multistage sorting into an integrated process that flexibly alternates between both methods to increase the traffic throughput of classification yards.

Chapter 4

Recovery Robust Train Classification

The improvement of the optimal algorithm for computing train classification schedule presented in Section 1.2.5 over the traditional sorting methods is partially based on taking the order of inbound trains into account. This order is subject to disturbance, and a computed schedule might become infeasible due to disruptions in the railway network that cause a different order than the expected one. The common sorting methods deal with this issue by completely ignoring the input order of cars, which makes them robust against any amount of disturbance but also wastes the mentioned potential contained in the knowledge about the input. This dilemma is tackled in this chapter by disregarding degrees of disruption that almost never occur in practice in order to obtain a classification process that is quicker than the sorting methods of the current railway practice but still provides robustness against realistic degrees of delay.

Section 4.1 first outlines several general concepts of robustness in optimization. Based on one specific of these notions, the precise problem setting of this chapter is introduced in the same section for distinct cars. In the context of the adapted notion of robustness, the approach of inserting additional sorting steps is then analyzed in detail in Section 4.2, for which an optimal algorithm for general delays is established in Section 4.3. For a more restricted but realistic class of delays, an efficient algorithm is developed, analyzed, and experimentally evaluated

in Section 4.4. Finally, the generalized problem setting with non-distinct cars is tackled in Section 4.5.

4.1 Introduction

First, different theoretical notions of robustness in optimization are outlined in Section 4.1. The precise robust setting for the train classification problem considered in this chapter is given in Section 4.1.2, followed by some generalized and additional notation in Section 4.1.3.

4.1.1 Notions of Robustness and Related Work

A lot of research has been done in the recent past on different notions of robustness [BS04, LLMS07, LLMS09, FM09, CDSS08] and their railway applications [LLMS07, CDD+08, LLMS09, CDD+07, CDSS08, CCG+08]

The concept of *strict robustness* suggests to deal with disruptions by preparing for any possible disruption in advance. Here, a so-called *robust* solution is calculated that is usually suboptimal for the case of no disruptions but feasible for any possible disruption scenario. This applies to the commonly applied classification methods presented in Section 1.1.3, including triangular and geometric sorting. They are robust w.r.t. any disturbance scenario since they apply a predefined classification schedule that is independent of the inbound order of cars entering the classification process. Put differently, these methods simply ignore any presortedness in the input and always assume a worst-case order.

Providing strict robustness, however, wastes a lot of potential to disruption scenarios that almost never occur in practice. A better solution for the undisturbed input can be achieved by regarding only realistic scenarios (similar to [BS04]) and, particularly, allowing some action to be taken after a disruption occurs. This leads to the concept of *recoverable robustness* [LLMS07, LLMS09]. Here, the (first-stage) solution may also be suboptimal if no disruption occurs, but it provides a computationally limited recovery action which, for any possible disturbance scenario, calculates a new solution that is feasible with respect to this scenario from the initial solution. This

approach is generalized to *dynamic recoverable robustness* in [CDSS08], a concept in which providing subsequent recovery steps is demanded so that multiple consecutive disruptions can be handled.

The concept of recoverable robustness is applied to different problems from railway optimization, such as rolling stock scheduling [CCG+08] or timetabling [CDSS08, CDD+09]. The first attempt to consider robustness questions in train classification is made in [CDD+07], which deals with several kinds of disruptions for the special case of a single incoming and a single outgoing train. (Their results are summarized in [CDD+09].) Besides the situations of strict robustness and complete recomputation from scratch, which are more of theoretical interest, the authors consider a recovery action that allows completely changing the assigned bitstrings for one set of cars that have the same bitstrings. The most relevant scenarios in [CDD+07] are one additional car in the input and one car occurring at a different position than expected. The latter corresponds to the problem setting of the special case of trains consisting of single cars in combination with the scenario of delaying up to one train. (The setting can also be regarded as the one of Section 4.5, i.e. singleton trains, but with distinct cars and delaying only one train.) A more general setting is introduced in Section 4.4 with more general trains and more general delay scenarios. Besides, [CDD+07] deals with the scenario of a single classification track becoming unavailable before the classification starts. In practice, however, unavailable tracks, faulty switches, and other disruptions with respect to the classification facilities only play a minor role. This chapter focuses on the most relevant reason for disruptions, which are delayed trains. The model of disruption is explained in further detail in the following section as a part of the problem setting.

4.1.2 Robust Problem Setting

As announced in Section 1.2.2 above, the order of inbound trains is not fixed anymore in this chapter. Different from the setting of Section 1.2.7, there still is an order of trains given in any problem instance. However, this presents only an expected order here, which is subject to disturbance by delays of trains.

The robust approach of this chapter follows the concept of

recoverable robustness introduced in the previous section. First, there is an expected order of inbound trains. The first-stage solution corresponds to a classification schedule that is feasible for the expected order of inbound trains. Second, there is a set of permutations of the inbound trains, each of which presents a possible disruption scenario. The modified instance, obtained by revealing a scenario from the set of possible scenarios, thus corresponds to a permutation of the expected order of trains, for which the first-stage schedule may be infeasible. Third, the means of recovery are given by inserting up to k additional sorting steps after the pth step of the first-stage schedule, where p and k are nonnegative integers. The schedule extended by k columns in this way presents the recovered solution. This model of recovery is defined formally and explained in more detail in Section 4.2.

Regarding the scenarios of disruption, the setting with general sets of scenarios is considered at first in Section 4.3. Handling such unrestricted sets of scenarios turns out to be impractical. A realistic class of scenarios that simply limits the number of delayed trains is then analyzed in Section 4.4. The precise definition is given in Section 4.4.1.

Throughout this chapter the cars of an instance are assumed to be distinct w.r.t. their types. An exception from this is made in Section 4.5, which considers the generalization to types of cars. The setting of this chapter also includes the case of multiple outbound trains.

Regarding the yard infrastructure, the classification tracks have no capacity restriction as mentioned in Section 1.2.2. There is no restricted number of sorting tracks either, and every sorting track involved in the classification process is pulled exactly once. Moreover, there is one output track for each of the outbound trains.

With all this, the problem dealt with in this chapter is defined as follows: given a sequence of ℓ inbound trains, m order specifications of outbound trains, a set of scenarios $\mathcal{S}$, and two recovery parameters p and k, the recovery-robust train classification problem is to find a feasible first-stage solution B of minimum length such that, for every scenario $S \in \mathcal{S}$, a feasible schedule for S can be obtained from B by inserting up to k additional columns left of the pth column.

4.1.3 Further Notation

Recall the concept of breaks which was introduced in Definition 3 for distinct cars without types: a pair of consecutive cars $(\tau, \tau{+}1)$ that belong to the same outbound train is called a break if $\tau{+}1$ occurs before τ in the inbound sequence of cars. Then, if the two cars also belong to the same inbound train, the pair is called an *internal* break, otherwise an *external* break. The concept of breaks is generalized in the definition below for the robust setting of this chapter.

Definition 4 *Let $\mathcal{T} = T_1, \ldots, T_\ell$ be an expected order of inbound trains yielding an inbound sequence of cars that is a permutation of $(1, \ldots, n)$. Let further $\mathcal{S}$ be a set of scenarios and $\beta = (\tau, \tau{+}1)$ be pair of consecutive cars of $\mathcal{T}$.*

β is called an original break *if β is a break for the expected order of inbound trains. For any $S \in \mathcal{S}$, β is called a break* induced by *S if it is not an original break but a break in the modified instance corresponding to S; the set of all breaks induced by S will be denoted by X^S. If $\beta \in X^S$ for some scenario $S \in \mathcal{S}$ but β is not an original break, β is called a* potential break.

W.l.o.g. every pair $\beta = (\tau, \tau{+}1)$, $\tau \in \{1, \ldots, n{-}1\}$, of successive cars in the same outbound train is assumed to either be an original or a potential break. (Otherwise, if β does not present a break, car $\tau{+}1$ can be ignored while deriving a schedule and assigned the same bitstring as τ in the final solution.) Referring to a break will henceforth mean an original or a potential break, so any pair $(\tau, \tau{+}1)$ presents a break. Further note that the set X^S is uniquely defined for every scenario $S \in \mathcal{S}$, but there may be different scenarios $S \neq S'$ with $X^S = X^{S'}$. Sets of potential breaks will repeatedly be regarded without considering the actual underlying scenario. In particular, sets of scenarios, e.g., when occurring as a parameter in a problem definition, will often be described implicitly by providing the set of induced breaks for every scenario.

Given any first-stage solution B and a scenario $S \in \mathcal{S}$, a break $\beta = (\tau, \tau + 1)$ is called *unresolved* w.r.t. S if β is induced by S and $b^\tau \geq b^{\tau+1}$. In other words, an unresolved break w.r.t. some scenario S will cause the first-stage solution to be infeasible for the modified instance corresponding to S because the order condition (1.3) will not be satisfied. Since only feasi-

ble first-stage classification schedules are considered, condition (1.2) always holds. Hence, the situation of $b^\tau > b^{\tau+1}$ never occurs, and $b^\tau = b^{\tau+1}$ can be assumed for every unresolved break $(\tau, \tau+1)$. Finally, let $T_1, \ldots, T_\ell$ be a sequence of ℓ inbound trains with n cars, and let X be the set of all (original and potential) breaks. For any pair of cars τ, σ, $1 \leq \tau < \sigma \leq n$, define $X_{(\tau,\sigma)}$ as the set of all breaks occurring between τ and σ, i.e., $X_{(\tau,\sigma)} = X \cap \{(\tau, \tau+1), (\tau+1, \tau+2), \ldots, (\sigma-1, \sigma)\}$.

4.2 Model of Recovery

Inserting a number of additional sorting steps after some offset of steps was outlined in Section 4.1.2 as the means of recovering a schedule, which will formally be defined in Definition 5. Why steps are added in the way that is specified in this definition is justified in Section 4.2.1. Section 4.2.3 finally relates a structural characteristic of robust schedules to the given set of scenarios. This immediately yields instructions for adding columns when a scenario is revealed. Moreover, this relation will be applied to find optimal recovery-robust solutions in Section 4.3.

4.2.1 Additional Sorting Steps

The common mode of operation of classification yards was explained in Section 1.1.3: a subset of classification tracks is reserved for multistage sorting, to which the involved cars, that arrive with different inbound trains scattered over the day, are continually rolled to the reserved tracks. Only when the last train has arrived, the process proceeds with secondary sorting. This means that the final order of inbound trains, i.e. the arising scenario, is only revealed when the (second) last train has arrived, i.e., shortly before the first pull-out operation is performed. The original schedule might be infeasible for this scenario. With the recovery action of inserting up to k additional sorting steps to the first-stage solution, a feasible schedule is sought for the modified instance. Distributing the recovered solution, i.e. the altered schedule, to all people involved in the operation takes some time depending on the available communication channels. For this reason inserting additional sorting steps is only allowed after an offset of p steps.

In terms of bitstring assignments, this means the following: given two parameters $p \geq 0$ and $k \geq 0$ and a first-stage schedule B of length h, B is to be recovered by inserting up to k additional columns left of column b_{p-1} of B. Essentially, adding columns will be applied in Lemma 11 to fix the order requirement (1.3) between cars of induced breaks that were unresolved in B.

Definition 5 *Let $B = b_{h-1} \dots b_0$ and $B' = b'_{h-1+k'} \dots b'_0$ be two classification schedules for n cars of length h and $h+k'$, $k' \geq 0$, respectively. Let further $p \geq 0$ and $k \geq 0$ be two nonnegative integers. Then, the schedule B' is called a* (p,k)-extension *of B if the following three conditions hold:*

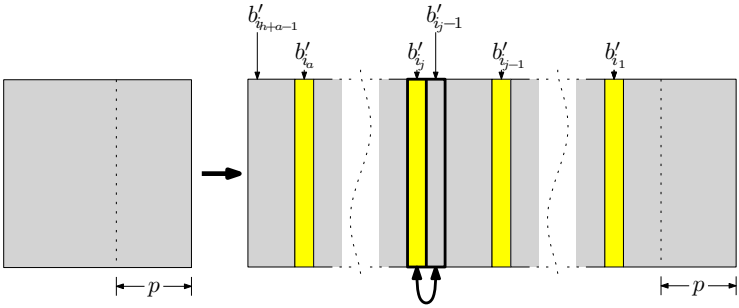
- $k' \leq k$,
- $b_i = b'_i$ *for all* $0 \leq i < p$, *and*
- $b_{i-k'} = b'_i$ *for all* $p+k' \leq i \leq h+k'-1$.

Note that in the definition above the additional columns are all added between the $(p-1)$th and pth step of the original schedule. This approach indeed presents the most powerful recovery as shown in Section 4.2.2 below, a result which is already covered in Definition 5.

The notion of (p,k)-extensions yields a natural notion of recovery-robust classification schedules, which is stated in the following definition.

Definition 6 *Let $T_1, \dots, T_\ell$ be a sequence of ℓ inbound trains and $\mathcal{S}$ be a set of scenarios. A classification schedule B is called* (p,k)-recovery-robust *if, for every scenario $S \in \mathcal{S}$, there exists a (p,k)-extension of B that is feasible for the modified sequence of inbound trains corresponding to S.*

For organizational reasons, applying any recovery in case no delay occurs, i.e., in case the inbound trains arrive in the expected order, is not wanted. This means it will always be a *feasible* (p,k)-recovery robust classification schedule of minimum length that is looked for.



(a) first-stage solution B (b) recovered schedule B'

Figure 4.1: The situation of Lemma 9, where B' is obtained from B by inserting a columns, but not necessarily between b_{p-1} and b_p.

4.2.2 Best Point for Additional Steps

A (p, k)-extension has just been defined as to insert up to k additional columns between the $(p-1)$th and pth step of the original schedule, which is the "right-most" allowed position. Lemma 10 further below shows that no other strategy to insert columns is more powerful than this. As an intermediate step, if B' is a recovered solution of B with inserted columns $b'_{i_1} \dots b'_{i_a}$, an inserted column B'_{i_j} can be swapped with its right neighbor b'_{i_j-1} if that is an original column of B. This is depicted in Figure 4.1 and formalized and proved in the following lemma.

Lemma 9 *Let $B = (b_{h-1} \dots b_0)$ be a feasible classification schedule of length h for n cars and $B' = (b'_{h-1+a} \dots b'_0)$ be a schedule for n cars of length $h + a$, $a \geq 0$, that is feasible w.r.t. some scenario $S \in \mathcal{S}$. Let further $I = \{i_1, \dots, i_a\}$ be a set of indices with $p \leq i_1 < i_2 < \dots < i_a \leq h + a - 1$ such that:*

- *$b'_k = b_k$ for $0 \leq k < i_1$,*
- *for every pair $i_j, i_{j+1} \in I$, $b'_k = b_{k-j}$ holds for every k with $i_j < k < i_{j+1}$, and*
- *$b'_x = b_{x-a}$ for $i_a < x$.*

Put $i_0 = p-1$; if there is a $j \in \{1, \dots, a\}$ such that $i_j - 1 > i_{j-1}$, then the schedule $B'' := b'_{h+a-1} \dots b'_{i_j+1} b'_{i_j-1} b'_{i_j} b'_{i_j-2} \dots b'_0$ is feasible w.r.t. S as well.

Proof Let $\tau < \sigma$ be two cars. If $b'^{\tau} = b'^{\sigma}$, then also $b''^{\tau} = b''^{\sigma}$ in the swapped schedule B''. Otherwise, $b'^{\tau} < b'^{\sigma}$ must hold as B' is feasible w.r.t. S. There are four sub-cases distinguishing between the length of the longest common prefix of b'^{τ} and b'^{σ}.

First, if $b'^{\tau}_{h+a-1} \dots b'^{\tau}_{i_j+1} < b'^{\sigma}_{h+a-1} \dots b'^{\sigma}_{i_j+1}$, then $b''^{\tau} < b''^{\sigma}$ still holds for B''. Second, if $b'^{\tau}_{h+a-1} \dots b'^{\tau}_{i_j+1} = b'^{\sigma}_{h+a-1} \dots b'^{\sigma}_{i_j+1}$ and $b'_{i_j} b'_{i_j-1}$ differ for τ and σ in exactly one bit, i.e., if they fulfil $(b'^{\tau}_{i_j} b'^{\tau}_{i_j-1}, b'^{\sigma}_{i_j} b'^{\sigma}_{i_j-1}) \in \{(00,10),(01,11),(00,01),(10,11)\}$, then $b'^{\tau}_{i_j-1} b'^{\tau}_{i_j} < b'^{\sigma}_{i_j-1} b'^{\sigma}_{i_j}$, so also $b''^{\tau} < b''^{\sigma}$. Third, if still $b'^{\tau}_{h+a-1} \dots b'^{\tau}_{i_j+1} = b'^{\sigma}_{h+a-1} \dots b'^{\sigma}_{i_j+1}$ but $b'_{i_j} b'_{i_j-1}$ differ in both bits, i.e. $b'^{\tau}_{i_j} b'^{\tau}_{i_j-1} = 01$ and $b'^{\sigma}_{i_j} b'^{\sigma}_{i_j-1} = 10$, then $b^{\tau} > b^{\sigma}$ held in the original schedule B. This contradicts the feasibility of B, so this case will not occur. Finally, if $b'^{\tau}_{h+a-1} \dots b'^{\tau}_{i_j+1} = b'^{\sigma}_{h+a-1} \dots b'^{\sigma}_{i_j+1}$ and even $b'^{\tau}_{i_j} b'^{\tau}_{i_j-1} = b'^{\sigma}_{i_j} b'^{\sigma}_{i_j-1}$, then $b''^{\tau} < b''^{\sigma}$ clearly holds after swapping. Summarizing, if $b'^{\tau} < b'^{\sigma}$, then $b''^{\tau} < b''^{\sigma}$ as well. □

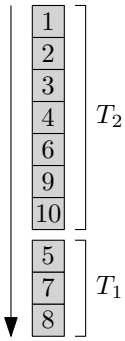
Applying the procedure of this proof repeatedly yields that no way of inserting columns presents a recovery that is superior to (p,k)-extensions. This is done in the following lemma.

Lemma 10 *Let $p \geq 0$ and $a \geq 0$ be nonnegative integers. Let further B, B', I, and $S \in \mathcal{S}$ be defined as in Lemma 9. Then, there is a (p,a)-extension of B that is feasible w.r.t. S.*

Proof If there is any $j \in \{1,\dots,a\}$ such that $i_j - 1 > i_{j-1}$, apply Lemma 9 to B'. Repeat this procedure to the respective resulting schedule with index set $I \setminus \{i_j\} \cup \{i_j - 1\}$ until it cannot be applied any more. Note that $i_j > i_j - 1 > i_{j-1}$, so the relative order of the elements in I never changes. Since a chosen column only moves to the right and the set of columns that are chosen is disjoint from the set of columns that are used to swap chosen columns with, this procedure eventually finishes. Then, for the schedule B' there is no $j \in \{1,\dots,a\}$ with $i_j - 1 > i_{j-1}$, so $i_j - 1 = i_{j-1}$ for every $j \in \{1,\dots,a\}$ and $i_1 = i_0 + 1 = p$. Therefore, B' is a (p,a)-extension that is feasible w.r.t. S. □

As a result, the search for a recovered schedule can always be restricted to looking for a feasible (p,k)-extension, which motivates Definition 6.

In fact, there are cases where there is a feasible (p,k)-recovery robust solution for which any other way of inserting columns fails to yield a feasible recovered solution for some scenario. This



(a) two inbound trains

b_2	b_1	b_0	car
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	0	6
1	0	1	7
1	1	0	8
1	1	0	9
1	1	1	10

(b) first-stage schedule B

b_2	b_1		b_0	car
0	0	0	0	1
0	0	0	1	2
0	1	0	0	3
0	1	0	1	4
1	0	0	0	5
1	0	1	0	6
1	0	1	1	7
1	1	0	0	8
1	1	1	0	9
1	1	1	1	10

(c) feasible (p, k)-extension B'

b_2		b_1	b_0	car
0		0	0	1
0		0	1	2
0		1	0	3
0		1	1	4
1	0	0	0	5
1	1	0	0	6
1	1	0	1	7
1	1	1	0	8
1	?	1	0	9
1		1	1	10

(d) attempt to insert left of b_1

Figure 4.2: Example classification problem with $p = 1$ and $k = 1$ for which any insertion strategy other than (p, k)-extensions is suboptimal.

is shown in the following example, which is also illustrated in Figure 4.2. Put $p = 1$ and $k = 1$. Let $T_1 = (8, 7, 5)$ be the first and $T_2 = (10, 9, 6, 4, 3, 2, 1)$ the second of two inbound trains to be classified (Figure 4.2(a)). Since there are seven original breaks, eight different bitstrings are required in any feasible solution, so the optimal length h is at least $h \geq \lceil \log_2 8 \rceil$. This value is indeed achieved by the schedule B shown in Figure 4.2(b), which is the only feasible schedule of length $h = 3$. For the simple scenario S that swaps the inbound order of T_1 and T_2, a $(1, 1)$-extension B' is shown in Figure 4.2(c). This schedule is feasible for the order corresponding to S: it resolves both induced breaks $(5, 6)$ and $(8, 9)$ and retains the order requirements (1.3) between all other pairs of consecutive cars, particularly $b'^8 > b'^7$. When trying to insert a column between b_1 and b_2 instead, the induced break $(5, 6)$ can only be resolved by inserting a 0-bit into b^5 and a 1-bit into b^6 as done in Figure 4.2(d). In order to retain the order requirements for $(6, 7)$ and for $(7, 8)$, a 1-bit must be inserted into b^7 and b^8 as well. However, condition (1.3) cannot be established between cars 8 and 9 then, so this strategy will fail. Exactly the same argument holds for inserting left of b_2, and there are no other positions for insert-

ing. Therefore, the shown $(1,1)$-extension is the only feasible recovered solution for the optimal initial solution.

As a result, (p,k)-extensions are the only optimal way among all recoveries that insert k additional sorting steps after an offset of p steps in general.

4.2.3 Responding to Revealed Scenarios

In order to specify when a given schedule is (p,k)-recovery robust for a given set of scenarios, the notion of a *block* of a schedule is introduced in the following definition. Basically, a block is defined as a maximal set of bitstrings representing integers between two powers of two.

Definition 7 *Let $p \geq 1$ be a nonnegative integer and B be a schedule of length h for an inbound sequence of n cars.*

For any bitstring b^τ of B, the substring $b^\tau_{h-1} \dots b^\tau_p$ is called the leading part *of b^τ, denoted by $b^\tau_{>p}$, and substring $b^\tau_{p-1} \dots b^\tau_0$ the* trailing part *of b^τ, denoted by $b^\tau_{<p}$. For any $\lambda \geq 0$, a subset of (not necessarily distinct) bitstrings $b^\tau, \dots, b^{\tau+\lambda-1}$ assigned to λ consecutive cars in B is called a* block *of B if their leading parts satisfy the following three conditions:*

- $b^{\tau-1}_{>p} < b^\tau_{>p}$,
- $b^\tau_{>p} = b^{\tau+y}_{>p}$ *for all* $1 \leq y \leq \lambda - 1$, *and*
- $b^{\tau+\lambda-1}_{>p} < b^{\tau+\lambda}_{>p}$.

Then, λ is called the size *of the block and car τ is called the* head (car) *of the block.*

Put differently, any block covers a maximal set of bitstrings that share a common leading part.

This concept is used in the following lemma to state the necessary and sufficient conditions to a schedule to have a (p,k)-extensions that is feasible w.r.t. some scenario. The second part of its proof presents a method to derive a (p,k)-extension for any given scenario S making use of the block structure of the first-stage schedule. The recovery is performed independently for every block, where unresolved breaks are successively fixed by raising the bitstring of the second car of the break and all cars following it up to the end of the block.

Lemma 11 *Let $T_1, \ldots, T_\ell$ be a sequence of ℓ inbound trains, B a feasible classification schedule, S a scenario, and $p, k \geq 0$. Then, there exists a (p,k)-extension of B that is feasible for the order of inbound trains corresponding to S if and only if the number of unresolved breaks w.r.t. S does not exceed $2^k - 1$ for any block of B.*

Proof ($\Rightarrow$) Let $X' = \{(\tau_1, \tau_1+1), \ldots, (\tau_t, \tau_t+1)\} \subseteq X^S$ be the set of unresolved breaks w.r.t. S within some block of B. In the following, for any nonnegative integer x, let $b(x)$ denote the binary representation of x. Assume, for contradiction, that $t > 2^k - 1$. For every $(\tau_i, \tau_i+1) \in X'$, schedule B satisfies $b^{\tau_i} = b^{\tau_i+1}$, so, for every (p,k)-extension $\bar{B} = \bar{b}_{h-1+k} \ldots \bar{b}_0$ of B, the leading part and the trailing part of $\bar{b}^{\tau_i}$ and $\bar{b}^{\tau_i+1}$ are equal. For $\bar{B}$ to be a valid schedule w.r.t. scenario S, every unresolved break (τ_i, τ_i+1), $i = 1, \ldots, t$, must therefore satisfy $\bar{b}^{\tau_i+1}_{p-1+k} \ldots \bar{b}^{\tau_i+1}_{p} > \bar{b}^{\tau_i}_{p-1+k} \ldots \bar{b}^{\tau_i}_{p}$. Since, for every $i = 1, \ldots, t-1$, $\bar{b}^{\tau_1}_{p-1+k} \ldots \bar{b}^{\tau_1}_{p} \geq b(0)$ and also $\bar{b}^{\tau_{i+1}}_{p-1+k} \ldots \bar{b}^{\tau_{i+1}}_{p} \geq \bar{b}^{\tau_i}_{p-1+k} \ldots \bar{b}^{\tau_i}_{p}$ must hold, the inequality $\bar{b}^{\tau_{t+1}}_{p-1+k} \ldots \bar{b}^{\tau_{t+1}}_{p} \geq b(t)$ is obtained, which is a contradiction since $t > 2^k - 1$. Hence, there is no (p,k)-extension of B that is valid w.r.t. S if $t > 2^k - 1$.

($\Leftarrow$) Let κ denote the number of blocks of B, $\bar{\tau}_i$ the head of the ith block, and λ_i the size of the ith block, $i = 1, \ldots, \kappa$. Let further $X'_i = \{(\tau^i_1, \tau^i_1+1), \ldots, (\tau^i_{t_i}, \tau^i_{t_i}+1)\} \subseteq X'$ be the set of unresolved breaks of the ith block with t_i denoting their respective number, $i = 1, \ldots, \kappa$.

The first-stage solution B is extended in the following way to a schedule $\bar{B}$: in the ith block, for every $j = 0, \ldots, t_i$, put $\bar{b}^{\tau}_{p-1+k} \ldots \bar{b}^{\tau}_{p} = [j]_2$ for every car τ with $\tau^i_j + 1 \leq \tau \leq \tau^i_{j+1}$, where $\tau^i_0 + 1 := \bar{\tau}_i$ is the head and $\tau^i_{t_i+1} := \bar{\tau}_i + \lambda_i - 1$ the last car of the ith block. If $|[j]_2| < k$ holds for the length of the bitstring, leading zeros can be added to obtain a bitstring of exact length k. Since $j \leq t_i \leq 2^k - 1$, also $|[j]_2| \leq k$, so the construction yields a (p,k)-extension of B.

It remains to show that this schedule $\bar{B}$ is feasible w.r.t. S. First, if $(\tau^i_j, \tau^i_j + 1) \in X'_i$ is an unresolved break, then $b^{\tau^i_j} = b^{\tau^i_j+1}$ in the original schedule B. The construction yields $\bar{b}^{\tau^i_j}_{p-1+k} \ldots \bar{b}^{\tau^j_i}_{p} = [j-1]_2 < [j]_2 = \bar{b}^{\tau^i_j+1}_{p-1+k} \ldots \bar{b}^{\tau^j_i}_{p}$, so $\bar{b}^{\tau^i_j} < \bar{b}^{\tau^i_j+1}$ holds for $\bar{B}$, which satisfies condition (1.3). Second, in case $(\tau, \tau+1) \in X^S \setminus X'$ is not unresolved, then $b^\tau < b^\tau + 1$ in B by

definition of unresolved breaks. If, in this case, b^τ and $b^\tau+1$ are in the same block of B, then $\bar{b}^\tau_{p-1+k}\dots\bar{b}^\tau_p = \bar{b}^{\tau+1}_{p-1+k}\dots\bar{b}^{\tau+1}_p$, so $\bar{b}^\tau < \bar{b}^{\tau+1}$ holds for $\bar{B}$. Otherwise, let b^τ be contained in the ith block and $b^\tau+1$ in the $(i+1)$th block; then, they satisfy $b^\tau_{>p} < b^{\tau+1}_{>p}$, so $\bar{b}^\tau < \bar{b}^{\tau+1}$ for $\bar{B}$. Finally, if $(\tau,\tau+1) \notin X^S$, i.e., the break is not induced by S, then $b^\tau \le b^{\tau+1}$ simply by the feasibility of B and (1.2). If then b^τ and $b^{\tau+1}$ are in the same block, $\bar{b}^\tau_{p-1+k}\dots\bar{b}^\tau_p = \bar{b}^{\tau+1}_{p-1+k}\dots\bar{b}^{\tau+1}_p$, so $\bar{b}^\tau \le \bar{b}^{\tau+1}$ still holds for the extension. Otherwise, let b^τ be contained in the ith and $b^{\tau+1}$ in the $(i+1)$th block, $i \in \{1,\dots,i_\kappa-1\}$; then, $b^\tau_{>p} < b^{\tau+1}_{>p}$ in the first-stage solution B, so $\bar{b}^\tau < \bar{b}^{\tau+1}$ holds for $\bar{B}$ as well. Hence, extension $\bar{B}$ is feasible for the order of inbound trains corresponding to S. □

The proof of this lemma provides a method for deriving a feasible (p,k)-extension from a first-stage schedule if such an extension exists. However, it does not yet show how a feasible recovery-robust first-stage solution can be found initially. This is dealt with in the following section.

4.3 General Scenarios

The observations of the previous section are applied to develop an algorithm for computing feasible (p,k)-recovery robust train classification schedules of minimum length in this section. Section 4.3.1 introduces a generic algorithm for general sets of scenarios. The complexity of this algorithm and of the problem in general is then analyzed in Section 4.3.2. The generic algorithm will be instantiated for a special set of scenarios in Section 4.4.

4.3.1 Optimal Algorithm

The algorithm for computing (p,k)-recovery robust schedules presented here follows a greedy strategy. It successively grows the size of a block until its maximum size is reached. The maximum size of a block is determined by two factors: first, any schedule B can use at most 2^p different trailing parts of bitstrings for cars in the same block. This simply follows from the definition of blocks and implies that at most 2^p-1 breaks can be resolved in the first-stage schedule, be it potential or original

breaks. Second, for every possible scenario, the number of unresolved breaks in a block of a recovery robust schedule must not exceed $2^k - 1$ by Lemma 11. This second condition is formalized in the following definition. Recall from the end of Section 4.1.3 that $X_{(\tau,\sigma)}$ denotes the set of all breaks occurring between the cars τ and σ.

Definition 8 *Let $T_1, \ldots, T_\ell$ be a sequence of ℓ inbound trains with a total of n cars and $\tau, \sigma \in \{1, \ldots, n\}$ be two cars. Let further $k \geq 0$ be a nonnegative integer.*

Then, given a set of scenarios $\mathcal{S}$, a set of breaks $X' \subseteq X_{(\tau,\sigma)}$ is called k-recoverable *w.r.t. to $X_{(\tau,\sigma)}$ if $|X' \cap X^S| \leq 2^k - 1$ holds for every scenario $S \in \mathcal{S}$.*

Algorithm 2 below greedily grows blocks one after another. While processing a block, it repeatedly solves the problem of finding a maximum k-recoverable set of breaks in order to increase the size of the block. It finishes with blocks of maximum size and will have constructed an optimal (p, k)-recovery robust schedule by this as shown further below.

In the pseudo code of Algorithm 2, the loop of line 4 iterates over the inbound sequence of cars, and a full execution of the loop in line 6 corresponds to completing a single block. The variables $\tau_1, \tau_2, \tau_3, \ldots$ will indicate the respective head cars of the blocks when the end of the inbound sequence is reached. τ_1 is initialized with car 1 in lines 1 and 2. While growing a block in an execution of the loop of line 6, variable $\tau_{\max}$ presents the temporary head of the subsequent block, X the set of all potential breaks in the temporary block, and X' a maximum k-recoverable subset of X. $\tau_{\max}$ is initialized with τ_1 in line 3 and X' with the empty set in line 5 so that the while loop in line 6 is initially entered unless $2^p \geq n$. In the latter case all cars can be covered by a single block. Whenever the xth block, $x \geq 1$, is started from head car τ_x, which corresponds to a first iteration of the while loop of line 6, the 2^p cars $\tau_x, \tau_x + 1, \ldots, \tau_x + 2^p - 1$ can directly be included in the block since 2^p different bitstrings are available. In all subsequent iterations, a maximum k-recoverable set of breaks X' is computed, and the block can be extended by the difference in size between X' and the corresponding set computed in the previous iteration. By the definition of a k-recoverable set of breaks, the

Algorithm 2: (p, k)-Recovery Robust Train Classification
Input: number of cars n, set of original breaks X_{org}, set of scenarios $\mathcal{S}$, recovery parameters k and p
Output: k-recovery robust classification schedule B

```
1  set x = 1
2  set τ_x = 1
3  set τ_max = τ_x
4  while (τ_x ≤ n) do
5    set X' = ∅
6    while (τ_max < τ_x + 2^p + |X'| and τ_x + 2^p + |X'| ≤ n) do
7      set τ_max = τ_x + 2^p + |X'|
8      set X = X_(τ_x, τ_max) ∩ ( ⋃_{S∈S} X^S )
9      compute max. k-recoverable set of breaks X' ⊆ X
10   set τ_max = min(τ_x + 2^p + |X'|, n + 1)
11   set τ_{x+1} = min(τ_x + 2^p + |X'|, n + 1)
12   compute subschedule of length p for τ_x, ..., τ_{x+1} − 1 in
     which all breaks of X_(τ_x, τ_{x+1}−1) \ X' are resolved
13   x++
14 put h' = ⌈log_2(x − 1)⌉
15 for (j = 0, ..., x − 1) do
16   for (τ = τ_x, ..., τ_{x+1} − 1) do
17     set b^τ_{p+h'−1} ... b^τ_p = [j]_2
18 return B
```

additional break can be handled in the recovery. In case of a negative evaluation of line 6, which indicates no increase in size has been achieved or the end of the inbound sequence has been reached, the block is finished and the head car of the next block is fixed in line 11. Furthermore, the trailing parts of the bitstrings of all cars in this block are computed (line 12) and the block index is incremented (line 13). When the last block is finished, i.e., when the check in the while loop of line 4 yields a negative result, the length of the schedule is determined according to the total number of blocks in line 14. The leading parts of all bitstrings are computed in lines 15 to 17 according to their block membership. The computed schedule B returned in line 18 is proved to be a (p, k)-recovery robust train classification schedule of minimum length in the following theorem.

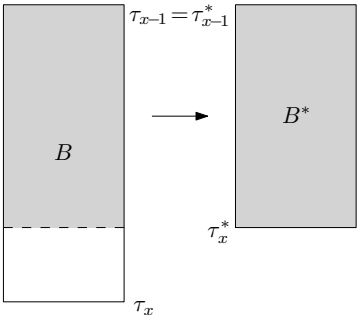
Theorem 8 *Given an inbound sequence of trains $\mathcal{T} = T_1, \ldots, T_\ell$ with a total of n cars, let X_{org} denote the corresponding set of original breaks. Let further $\mathcal{S}$ be a set of scenarios and $p, k \geq 0$ nonnegative integers.*

The schedule B returned by Algorithm 2 for the input n, X_{org}, $\mathcal{S}$, k, and p presents an optimal (p,k)-recovery robust train classification schedule.

Proof Let κ denote the number of blocks in B, so $\tau_1, \ldots, \tau_\kappa$ are the heads of the blocks. The unresolved breaks of any block of B present a k-recoverable set of breaks by construction and all original breaks are resolved in B by line 12. It follows that B is a feasible (p,k)-recovery robust classification schedule.

Assume, for contradiction, that B is not optimal. Then, there exists a (p,k)-recovery robust schedule B^* the length of which is shorter than that of B. Let $\kappa^* < \kappa$ be the number of blocks of B^*, and let $\tau_1^*, \ldots, \tau_{\kappa^*}^*$ be their respective heads. Define $\tau_{\kappa+1} = \tau_{\kappa^*+1}^* = n+1$ and let $\tau_x \neq \tau_x^*$ for some $x \in \{0, \ldots, \kappa\}$ with $\tau_z = \tau_z^*$ for all $z < x$. W.l.o.g. assume B^* is chosen such that it maximizes the value of x. There are two cases.

Case 1: $\tau_x > \tau_x^*$. If the bitstrings $b^{*\tau_{x-1}^*}, \ldots, b^{*\tau_x}$ in schedule B^* are replaced by the bitstrings $b^{\tau_{x-1}^*}, \ldots, b^{\tau_x}$ of B, the result is an optimal schedule in which the head of the xth block is given by τ_x. Since $\tau_x > \tau_x^*$, this contradicts the choice of B^*.



Case 2: $\tau_x < \tau_x^*$. Let Y_{x-1}^* be the set of unresolved breaks in the $(x-1)$th block of B^* and Y_{x-1} the set of unresolved breaks in the $(x-1)$th block of B. Since Y_{x-1} is a maximum k-recoverable set of breaks by the construction of B in Algorithm 2, its size satisfies $|Y_{x-1}| \geq |Y_{x-1}^*|$. Consider the number of resolved breaks of B^* in its $(x-1)$th block:

$$\begin{aligned} |X_{(\tau_{x-1},\tau_x^*)}| - |Y_{x-1}^*| &\geq |X_{(\tau_{x-1},\tau_x^*)}| - |Y_{x-1}| \\ &\overset{(*)}{>} |X_{(\tau_{x-1},\tau_x)}| - |Y_{x-1}| \\ &= 2^p - 1 \end{aligned}$$

(Recall that every pair of consecutive cars is assumed to be either a potential or an original break, which is why inequality $(*)$ holds.) This is a contradiction since there are not more than 2^p different bitstrings available for a block. Therefore, such a schedule B^* does not exist, so B is a shortest (p,k)-recovery robust schedule. □

Algorithm 2 thus yields an optimal recovery robust schedule, but the actual step of computing a maximum k-recoverable set of breaks in line 9 is not specified explicitly. The k-recoverable set of breaks can, for example, be calculated by integer programming as follows. For every break $\beta \in X = \bigcup_{S\in\mathcal{S}} X^S \cap X_{(\tau_x,\tau_{\max})}$, there is a binary variable x_β indicating whether β is a member of the k-recoverable set of breaks ($x_\beta = 1$) or not ($x_\beta = 0$). With this, any optimal solution of the following integer programming model presents a maximum k-recoverable set of breaks:

$$\begin{aligned} \max \quad & \sum_{\beta\in X} x_\beta \\ & \sum_{\beta\in X^S\cap X_{(\tau_x,\tau_{\max})}} x_\beta \le 2^k - 1, \quad S \in \mathcal{S} \\ & x_\beta \in \{0,1\}, \quad \beta \in X \end{aligned}$$

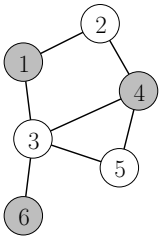
In general, however, finding a maximum k-recoverable set of breaks presents an $\mathcal{NP}$-hard optimization problem as shown in the following section. Unfortunately, also the problem of finding a (p,k)-recovery robust train classification schedule turns out to be $\mathcal{NP}$-hard.

4.3.2 Computational Complexity

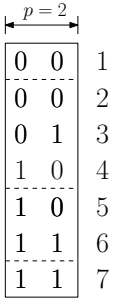
In Algorithm 2 a maximum k-recoverable set of breaks X' is repeatedly computed for the set X of all original and potential breaks of the block currently grown. W.l.o.g. assume that $X = \{1,\dots,n\}$, which means, given a set of scenarios $\mathcal{S}$, to look for a maximum k-recoverable break set X' of $X = \bigcup_{S\in\mathcal{S}} X^S$. In the following theorem, this problem is shown to be strongly $\mathcal{NP}$-hard. For $k = 1$ this is done by a reduction from the independent set problem to the decision version of this problem, for $k \ge 2$ a reduction from 2^kSAT is used.

Theorem 9 *Let $T_1, \ldots, T_\ell$ be a sequence of ℓ inbound trains, $\mathcal{S}$ be a set of scenarios, and $K \geq 0$. For every constant $k \geq 1$, deciding whether there exists a k-recoverable set of breaks of size K presents a strongly* **NP***-hard problem.*

Proof ($k = 1$) A polynomial-time reduction from the independent set problem, which is known to be $\mathcal{NP}$-hard [GJ79a], is applied in the following. Let $G = (V, E)$, $V = \{1, \ldots, n\}$, be a graph in which an independent set of size a is sought. W.l.o.g. assume G to contain no isolated vertex and $|V| - a = 2^p - 1$.



In the corresponding instance of the problem of finding a 1-recoverable set of breaks there will be $n+1$ cars $1, \ldots, n+1$. Furthermore, for every edge $(x, y) \in E$, the set of scenarios will contain a scenario S that induces the set of breaks $X^S = \{(x, x+1), (y, y+1)\}$. Note that the size of $X = \cup_{S \in \mathcal{S}} X^S$ and the number of scenarios $\mathcal{S}$ is polynomial in the size of G. It remains to show that there is a recoverable set of breaks $X' \subseteq X$ of size a if and only if there is an independent set of size a in G.
Let A be an independent set in G with $|A| = a$. Define the set of breaks $X_A = \{(x, x+1) \mid x \in A\}$. For any scenario $S \in \mathcal{S}$ with $X^S = \{(x, x+1), (y, y+1)\}$, there is an edge $\{x, y\} \in E$, and either $x \notin A$ or $y \notin A$. Thus, at most one of the breaks of X^S is an element of X_A, so X_A is a 1-recoverable break set of size a.



Conversely, let X' be a 1-recoverable set of breaks of size a. Define $A = \{x \mid (x, x+1) \in X'\}$. For any edge $\{x, y\} \in E$ there is a scenario $S \in \mathcal{S}$ with $X^S = \{(x, x+1), (y, y+1)\}$. Since at most one break can be recovered for $k = 1$, either $(x, x+1) \notin X'$ or $(y, y+1) \notin X'$, so at most one endpoint of the edge is in A. Therefore, A is an independent set of size a.

($k > 1$) For the case $k > 1$ a polynomial-time reduction from 2^kSAT is applied, which is strongly $\mathcal{NP}$-hard too [GJ79b]. Let I be an instance of 2^kSAT with variables $x_1, \ldots, x_n$ and clauses $\mathcal{C} = \{C_1, \ldots, C_m\}$. Each clause contains 2^k different literals. For each variable x_i add the auxiliary variables $x_{1,i}, \ldots, x_{2^{k-1},i}$ and a clause $\bar{C}_i = x_i \vee \bar{x}_i \vee x_{1,i} \vee \bar{x}_{1,i} \vee \ldots \vee x_{2^{k-1}-1,i} \vee \bar{x}_{2^{k-1}-1,i}$

to the instance. As every added clause $\bar{C}_i$ presents a tautology, there exists an satisfying assignment for the modified instance if and only if there is one for the original instance.

In the instance of the problem of finding a k-recoverable set of breaks, let there be $2n+n(2^k-2)+1$ cars, which corresponds to $2n+n(2^k-2)$ possible breaks. As this number matches the number of literals, there is a bijection between the set of all breaks and the set of literals. For each clause C_j, add a scenario S_j in which $\beta_i \in X^{S_j}$ iff $x_i \in C_j$ and $\bar{\beta}_i \in X^{S_j}$ iff $\bar{x}_i \in C_j$. In the same way, for each clause $\bar{C}_j$, define a scenario $\bar{S}_j$. Hence, there are $m+n$ scenarios of size 2^k each and $2n+n(2^k-2)$ breaks. Since k is constant, the size of the instance is polynomial in the size of the input data. It remains to show there exists a k-recoverable set of breaks $X' \subseteq \bigcup_{S\in\mathcal{S}} X^S$ with exactly $n+n(2^k-2)$ breaks if and only if there is a satisfying assignment for I.

Let X' be a k-recoverable set of breaks with cardinality $|X'| = n+n(2^k-2)$. W.l.o.g. assume all breaks $\beta_{i,j}$ and $\bar{\beta}_{i,j}$ are contained in X'. (Otherwise, replace β_i by $\beta_{i,j}$ or $\bar{\beta}_i$ by $\bar{\beta}_{i,j}$ and obtain a k-recoverable set of breaks again. This can be iteratively applied until all $\beta_{i,j}$ and $\bar{\beta}_{i,j}$ are contained: if there were two breaks $\beta_{i,j}$ and $\beta_{i,j'}$ for some $i \in \{1,\dots,n\}$, there would be some $i' \in \{1,\dots,n\}$ for which *all* the 2^k literals of $C_{i'}$ are contained in X' by the pigeonhole principle; then, however, scenario $\bar{S}_{i'}$ would induce 2^k unresolved breaks and X' would not be k-recoverable.) With $|\bigcup_{S\in\mathcal{S}} X^S| - |X'| = n$ and the definition of the clauses $\bar{C}_i$, the set X' contains either β_i or $\bar{\beta}_i$ but never both. Thus, the following assignment is well-defined: $x_i^* = 1$ if $\bar{\beta}_i \in X'$ and $x_i^* = 0$ if $\beta_i \in X'$. Since every S_j contains exactly 2^k breaks, at least one break in X^{S_j} is resolved, i.e., there is a break in X^{S_j} which is not in X'. If this break is of the type $\beta_\alpha \in X^{S_j}$, clause C_j contains literal x_α and $x_\alpha^* = 1$ verifies C_j. If $\bar{\beta}_\alpha \in X^{S_j}$ and it is resolved, the clause C_j contains the literal $\bar{x}_\alpha$ and $x_\alpha^* = 0$. Hence, x^* verifies every clause.

Conversely, let x^* be a satisfying assignment for the 2^kSAT-instance I. If $x_i^* = 1$, delete β_i from $\bigcup_{S\in\mathcal{S}} X^S$, and, if $x_i^* = 0$, delete $\bar{\beta}_i$, to construct a maximum k-recoverable set of breaks X'. The set X' contains $n+n(2^k-2)$ elements. Furthermore, in every scenario S_j at most 2^k-1 breaks are unresolved. Therefore, X' is a k-recoverable set of breaks. □

As a direct consequence from the above theorem, Algorithm 2 will only run in polynomial time if **P** = **NP**. Anyway, finding an optimal (p, k)-recovery robust classification schedule is $\mathcal{NP}$-hard as shown in the following corollary.

Corollary 1 *Let $T_1, \ldots, T_\ell$ be a sequence of ℓ inbound trains, $\mathcal{S}$ be a set of scenarios, $h, p \geq 0$ be two nonnegative integers, and $k \geq 1$ be a positive constant.*

Deciding whether there exists a feasible (p, k)-recovery robust classification schedule of length at most h is an $\mathcal{NP}$-hard problem.

Proof The decision version of the problem of finding a maximum k-recoverable set of breaks can be reduced to the stated problem in polynomial time by the following construction. For the given set of scenarios $\mathcal{S}$, define $X = \bigcup_{S \in \mathcal{S}} X^S$, and let $K \geq 0$ be some constant. W.l.o.g. assume $|X| - K = 2^h - 1$ for some $h \geq 0$. (Otherwise, put $\Delta = (2^h - 1) - (|X| - K)$ and add Δ more breaks to the instance. Also add every such break to every scenario $S \in \mathcal{S}$. Then, there exists a k-recoverable set of breaks of size K in the original instance if and only if there exists one of size $K - \Delta = |X| - (2^h - 1)$ in the modified instance.) Then, there exists a feasible (h, k)-recovery robust schedule of length at most h for this instance if and only if there is a k-recoverable set of breaks $X' \subseteq X$ of size K. □

In the statement of the above corollary, there are no restrictions for the size or structure of the set of scenarios $\mathcal{S}$. Hence, there may still be sets of scenarios for which an efficient algorithm can be devised. A class of such sets is presented in Section 4.4 below, for which an efficient algorithm is devised based on Algorithm 2.

4.4 Limited Numbers of Delay

As mentioned before, providing strict robustness wastes a lot of potential to extreme scenarios that rarely occur. For this reason a simple yet general class of scenarios is introduced in this section.

4.4.1 Scenario Model

Loosely speaking, the class of scenarios considered here is given by limiting the number of delayed trains but allowing an arbitrary degree for each single delay. The number of delayed trains will be given by some integer parameter $j \geq 0$.

More precisely, let $\mathcal{T} = T_1, \ldots, T_\ell$ be an inbound sequence of trains and $\mathcal{T}^\pi = T_{\pi^{-1}(1)}, \ldots, T_{\pi^{-1}(\ell)}$ be an order of trains induced by some permutation $\pi\colon \{1, \ldots, \ell\} \to \{1, \ldots, \ell\}$. Then, a sequence of trains $\mathcal{T}^{\bar{\pi}} = T_{\bar{\pi}^{-1}(1)}, \ldots, T_{\bar{\pi}^{-1}(\ell)}$, for $\bar{\pi}$ being a further permutation, will be called an (α, k)-*delayed sequence* of $\mathcal{T}^\pi$ if $k \geq \pi(\alpha)$ and the following three conditions hold:

- $\bar{\pi}(x) = \pi(x)$ if either $\pi(x) < \pi(\alpha)$ or $\pi(x) > k$,
- $\bar{\pi}(x) = \pi(x) - 1$ for $\pi(\alpha) < \pi(x) < k$, and
- $\bar{\pi}(\alpha) = k$.

Less formally, train T_α of the original sequence is delayed from its position $\pi(\alpha)$ in $\mathcal{T}^\pi$ to the kth position in $\mathcal{T}^{\bar{\pi}}$, and no other train is delayed. With this, the class of scenarios dealt with in the remainder can now be defined by applying a sequence of such single delays.

Definition 9 *For a sequence of ℓ inbound trains $\mathcal{T} = T_1, \ldots, T_\ell$, let $\mathcal{T}^S$ denote the sequence of inbound trains induced by some scenario S. Let further j be an integer with $0 \leq j \leq \ell$.*

The set of scenarios $\mathcal{S}_j$ is defined to contain the scenario S if and only if there is a sequence $\mathcal{T}^0, \ldots, \mathcal{T}^j$ of train sequences such that the following three conditions hold:

- $\mathcal{T}^0 = \mathcal{T}$,
- *for every $i = 1, \ldots, j$, there are values $\alpha, k \in \{1, \ldots, \ell\}$ for which $\mathcal{T}^i$ is an (α, k)-delayed sequence of $\mathcal{T}^{i-1}$, and*
- $\mathcal{T}^j = \mathcal{T}^S$.

In this case, every train T_{α_i} will be called to be delayed *by S.*

Notice the application of *exactly* j single delay steps in the above definition. For any permutation π, the sequence of trains $\mathcal{T}^\pi$ is a $(1, \pi(1))$-delayed sequence of itself, which corresponds to a

zero-delay of train T_1. Hence, the above definition does also cover scenarios in which strictly less than j trains are delayed as announced above. However, it suffices to restrict the consideration to a subset of the scenarios with exactly j delayed trains as shown in the following section.

4.4.2 Dominating Set of Scenarios

The set of scenarios $\mathcal{S}_j$ defined in the previous section contains all scenarios that affect up to j trains by an arbitrary delay each. Intuitively, a scenario with less than j delayed trains cannot present the worst case. Even if exactly j trains are affected, the worst case disruption should not be achieved for minor delays of these trains. Indeed, it suffices to regard a subset of scenarios of $\mathcal{S}_j$ which all delay the full j trains and each such train by the highest possible amount. This set $\bar{\mathcal{S}}_j \subseteq \mathcal{S}_j$ of scenarios is defined as follows and shown to dominate $\mathcal{S}_j$ in Theorem 10 further below.

Definition 10 *For an inbound sequence of trains $\mathcal{T} = T_1, \ldots, T_\ell$ of a classification instance, let S be a scenario and $\mathcal{T}^S$ denote the sequence of inbound trains induced by S. Let further j be an integer with $0 \leq j \leq \ell$. Then, S is a member of $\bar{\mathcal{S}}_j$ if and only if there is a sequence $\mathcal{T}^0, \ldots, \mathcal{T}^j$ of train sequences and a corresponding sequence of indices $\alpha_i \in \{1, \ldots, \ell\}$, $i = 1, \ldots, j$, for which the following four conditions hold:*

- $\mathcal{T}^0 = \mathcal{T}$,
- *for all $i = 1, \ldots, j$, $\mathcal{T}^i$ is an (α_i, ℓ)-delayed sequence of $\mathcal{T}^{i-1}$,*
- $\alpha_i < \alpha_{i-1}$ *for all* $i = 1, \ldots, j$, *and*
- $\mathcal{T}^j = \mathcal{T}^S$.

When a scenario is constructed by j sequential delays according to this definition, the delays are processed in reverse order of the scheduled arrivals of the trains, i.e., the construction processes trains with a high index first, and each train is delayed to the end of the inbound sequence of trains. In this way, if two trains are delayed by some scenario $S \in \bar{\mathcal{S}}_j$, they will swap their relative order and arrive later than all punctual trains. Note that for uniquely defining a scenario $S \in \bar{\mathcal{S}}_j$ it suffices to list

the j delayed trains since the order and amount of their delay is determined by the definition of $\bar{S}_j$.

Theorem 10 *For an inbound sequence of trains $\mathcal{T} = T_1, \ldots, T_\ell$ of a classification instance and some integers $0 \leq j \leq \ell$, $p, k \geq 0$, let B be a feasible (p,k)-recovery robust schedule for $\bar{\mathcal{S}}_j$. Then, B is a feasible (p,k)-recovery robust schedule for $\mathcal{S}_j$ as well.*

Proof The set of original breaks of $\mathcal{T}$ is independent of the set of scenarios. As B is feasible, all original breaks are resolved in B, so it clearly is a feasible schedule for $\mathcal{S}_j$.

Let $S \in \mathcal{S}_j$ be some scenario, and let $T_{i_1}, \ldots, T_{i_j}$ denote the trains delayed by S. Consider the scenario $\bar{S} \in \bar{\mathcal{S}}_j$ uniquely defined by the trains $T_{i_1}, \ldots, T_{i_j}$, and let $\bar{\pi}$ denote the permutation of the inbound trains corresponding to $\bar{S}$. Now, let $\beta = (\tau, \tau{+}1) \in X^S$ be a potential break induced by S.

Then, $\tau \in T_x$ for some $x \in \{i_1, \ldots, i_j\}$ and $\tau{+}1 \in T_y$ for some $x < y \leq \ell$. If T_y is among the trains delayed by $\bar{S}$, i.e. $y \in \{i_1, \ldots, i_j\}$, then $\bar{\pi}(x) > \bar{\pi}(y)$ since $x < y$; if $y \notin \{i_1, \ldots, i_j\}$, then $\bar{\pi}(y) \leq \ell - j < \bar{\pi}(x)$. In either case $\bar{\pi}(x) > \bar{\pi}(y)$, which means $\beta \in X^{\bar{S}}$, so $X^S \subseteq X^{\bar{S}}$. Therefore, schedule B is recovery robust also for $\mathcal{S}_j$. □

It is easy to see that any potential break $(\tau, \tau{+}1)$ can only be induced by S if the train containing τ is delayed. (A formal proof is given by the second half of the proof of Lemma 12.) The converse implication does not necessarily hold for scenarios $S \in \mathcal{S}_j$ as can be seen by the following simple example: if $\beta = (\tau, \tau{+}1)$ is a potential break with $\tau \in T_1$ and $\tau{+}1 \in T_2$, let S be a scenario delaying T_2 to, say, the end of the sequence of trains and T_1 to just one position before T_2; for $j \geq 2$, clearly $S \in \mathcal{S}_j$, but β will not be induced by S since T_1 and T_2 do not swap their relative positions.

This scenario S is not a member of $\bar{\mathcal{S}}_j$ for any j. In fact, the break above will be induced by *every* scenario of $\bar{\mathcal{S}}_j$ that delays T_1. This fact is stated in the following lemma, which will be used in the subsequent section to find maximum recoverable sets of breaks for $\mathcal{S}_j$ efficiently.

Lemma 12 *Let $T_1, \ldots, T_\ell$ be a sequence of inbound trains, and let $S \in \bar{S}_j$ be some scenario. For any potential break $\beta = (\tau, \tau{+}1)$ with $\tau \in T_x$, $x \in \{1, \ldots, \ell\}$, the break satisfies $\beta \in X^S$ if and only if T_x is delayed by S.*

Proof Let π be the permutation defined by S, and let $\tau{+}1 \in T_y$ for some $x < y \leq \ell$. If T_x is delayed and T_y as well, $x < y$ yields $\pi(x) > \pi(y)$ by the definition of $\bar{\mathcal{S}}_j$; if T_x is delayed but T_y is not, then $\pi(x) > \ell - j \geq \pi(y)$. In either case, delaying T_i will induce the break β.

Conversely, if T_x is not delayed but T_y is, $\pi(x) \leq \ell - j < \pi(y)$ and β is not induced; if neither T_x nor T_y are delayed, the trains cannot change their relative order, so β is not induced. Hence, β is only induced if T_x is delayed. □

As a consequence, if some inbound train T_x is delayed by some scenario $S \in \bar{\mathcal{S}}_j$, *all* potential breaks $\beta = (\tau, \tau{+}1)$ with $\tau \in T_x$ will necessarily be introduced. This fact is applied in the following section.

4.4.3 Maximum Recoverable Sets of Breaks

According to Lemma 12 the set of potential breaks X^S of any scenario $S \in \bar{\mathcal{S}}_j$ can be partitioned into disjoint subsets w.r.t. the respective train causing the break. For every $i = 1, \ldots, \ell$, the respective partition $X_i := \{(\tau, \tau{+}1) \mid \tau \in T_i, \exists y > i : \tau{+}1 \in T_y\}$ of X^S will be called the *set of breaks induced by* T_i. This partitioning serves as an input to Algorithm 3, which computes a recoverable set of breaks for the set of scenarios $\bar{\mathcal{S}}_j$ and thus $\mathcal{S}_j$. The algorithm repeatedly resolves potential breaks of the train that induces the highest number of unresolved breaks until the worst case scenario does not exceed the recovery capability given by the parameter k.
The worst case scenario of $\bar{\mathcal{S}}_j$ is presented by delaying the inbound trains with the j highest numbers of unresolved breaks. To estimate this number and keep track of it during the execution, the partitions of the potential breaks are sorted according to their size in line 1, and the variable α is initialized with the index of a train inducing the highest number of unresolved breaks. While the worst case cannot be handled, which is checked in line 3, it is reduced by excluding a break of T_α from the solution in line 4 and updating the value of α.

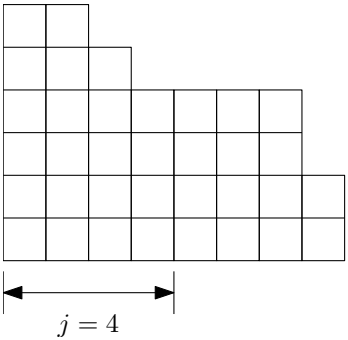
An example run of Algorithm 3 is given in Figure 4.3 for $j = 4$ and a recovery parameter value of $k = 4$, which corresponds to at most $2^k - 1 = 15$ unresolved breaks allowed in any delay scenario. In the example there are $\ell = 8$ inbound trains,

Algorithm 3: Maximum Recoverable Set of Breaks
Input: parameters $j, k \geq 0$, sets of breaks $X_1, \ldots, X_\ell$
Output: maximum k-recoverable subset of breaks of $\bigcup_{i=1}^{\ell} X_i$

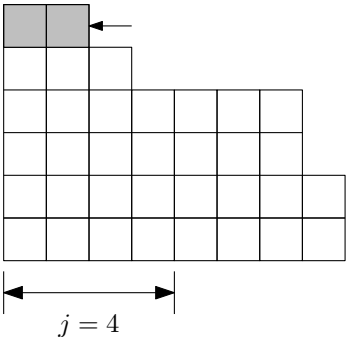
1 descendingly sort $X_1, \ldots, X_\ell$ such that $|X_{i_1}| \geq \ldots \geq |X_{i_\ell}|$
2 put $\alpha := \max_{t=1\,..\,\ell} \{i_t : |X_{i_t}| = |X_{i_1}|\}$
3 **while** $\left(\sum_{t=1}^{j} |X_{i_t}| \geq 2^k \right)$
4 remove an arbitrary break from X_{i_α}
5 put $\alpha := \max_{t=1\,..\,\ell} \{i_t : |X_{i_t}| = |X_{i_1}|\}$
6 **return** $\bigcup_{i=1}^{\ell} X_i$

each inducing between two and six breaks, which yield eight disjoint partitions of potential breaks, and the worst case scenario induces 21 of the potential breaks. In the first iteration of the while-loop of line 3, a break is removed from one of the two trains with six breaks, and from the other in the second iteration, which results in the situation shown in Figure 4.3(b). The following three iterations remove one break each from the three partitions with seven remaining unresolved breaks, which yields a worst case scenario of 16 unresolved breaks in Figure 4.3(c). There are seven trains now with four unresolved breaks each. After removing one break each from four of these trains, the condition of line 3 is not satisfied any longer, which means that the potential breaks not removed up to here together present a k-recoverable set of breaks. This final situation is depicted in Figure 4.3(d).

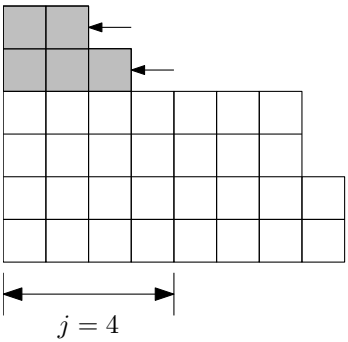
Observe the definition of α in the two respective lines of Algorithm 3, which always corresponds to the *highest* index among all trains currently inducing the highest number of unresolved breaks. In this way, the order previously computed in line 1 remains invariant during the execution of the while-loop of line 3. In Figure 4.3 this corresponds to cutting the peak line of breaks in the order from right to left. It furthermore causes all columns that have been cut to differ by at most one box. More formally, if two sets X_{i_a} and X_{i_b} with $1 \leq a < b \leq \ell$ have been reduced by at least one break each, then either $|X_{i_b}| = |X_{i_a}| - 1$ or $|X_{i_b}| = |X_{i_a}|$, which implies $|X_{i_b}| \geq |X_{i_a}| - 1$. These observa-



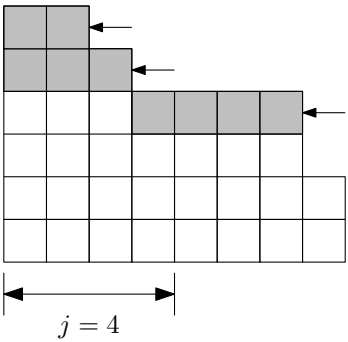
(a) The biggest j numbers of breaks sum up to 21 initially.



(b) After two iterations of line 3, the worst-case scenario introduces 19 of the potential breaks.



(c) After three more iterations, the maximum amounts to 16—still too much.



(d) Only after another four iterations, not more than 15 additional breaks may occur for any scenario.

Figure 4.3: An example run of Algorithm 3 for $j = 4$ delayed trains in the worst case and a recovery parameter value of $k = 4$. Columns correspond to outbound trains, boxes to single potential breaks in outbound trains, grey boxes to resolved breaks.

tions will be used in the proof of the following theorem, which shows that Algorithm 3 actually computes a maximum recoverable set of breaks. The analysis also includes the algorithm's correctness and running time.

Theorem 11 *Given a set of potential breaks X for some classification instance with inbound trains $T_1, \ldots, T_\ell$, a maximum k-recoverable set of breaks $\bar{X} \subseteq X$ w.r.t. $\bar{\mathcal{S}}_j$ can be computed in polynomial time.*

Proof Let $Y_i := X_i \cap X$ be the subset of breaks induced by T_i, $i = 1, \ldots, \ell$, and let $\bar{X} = \bigcup_{i=1}^{\ell} X'_i$ be the set returned by Algorithm 3 for the input $Y_1, \ldots, Y_\ell$.

In order to show that $\bar{X}$ is a feasible solution, let $S \in \bar{S}_j$ be a scenario delaying some inbound trains $T_{i_1}, \ldots, T_{i_j}$, i.e., S induces the set of breaks $Y^S := \bigcup_{t=1}^{j} Y_{i_t}$. The number of unresolved breaks induced by this scenario S satisfies the following chain of inequalities:

$$|\bar{X} \cap Y^S| = \sum_{t=1}^{j} |\bar{X} \cap Y_{i_t}| = \sum_{t=1}^{j} |X'_{i_t}| \overset{(*)}{\leq} \sum_{t=1}^{j} |X'_{i_t}| \leq 2^k - 1.$$

Note that inequality $(*)$ holds since the ordering defined in line 1 is invariant. Therefore, the recovery capabilities suffice to recover all unresolved breaks of X induced by S, so X' is k-recoverable.

In order to show that $\bar{X}$ also is an optimal solution, define the subset $X' = \bigcup_{t=1}^{j} X'_{i_t} \subseteq \bar{X}$. If the while-loop is entered at all, $|X'|$ decreases by exactly one in every iteration of the while-loop until $|X'| \leq 2^k - 1$, so $|X'| = 2^k - 1$ holds at the time the while-loop is left. Therefore, in case $|X'| < 2^k - 1$, the while-loop of line 3 has never been entered. This means $|X| < 2^k - 1$, so $X' = X$ and X' is optimal.

Otherwise, $|X'| = 2^k - 1$ as soon as the condition in line 3 is fulfilled. Then, every set X'_{i_t} with $t \geq j$ that has been decreased at least once satisfies $|X'_{i_t}| = |X'_{i_j}|$. For this case, assume, for contradiction, that there is some k-recoverable set $Z \subseteq X$ with $|Z| > |\bar{X}|$ and analogously define subsets of the partitions $Z_i := Y_i \cap Z$, $i = 1, \ldots, \ell$. Consider the order $|X'_{i_1}| \geq \ldots \geq |X'_{i_\ell}|$ in line 1 of Algorithm 3 for these subsets. Since Z is k-recoverable, $\sum_{t=1}^{j} |Z_{i_t}| \leq 2^k - 1$.

Since Z is k-recoverable, $\sum_{t=1}^{j} |Z_{i_t}| \leq 2^k - 1$. Equality $\sum_{t=1}^{j} |Z_{i_t}| = 2^k - 1$ can be assumed w.l.o.g. as shown in this paragraph. Otherwise, there are indices $a \leq j$ and $b > j$ with $|X'_{i_a}| > |Z_{i_a}|$ and $|X'_{i_b}| < |Z_{i_b}|$. This means at least one break has been removed from X'_{i_b} compared to the initial partition X_{i_b}, in which case $|X'_{i_b}| \geq |X'_{i_a}| - 1$. Define $\hat{Z}_{i_a} := Z_{i_a} + \beta_1$ for any $\beta_1 \in X'_{i_a} \setminus Z_{i_a}$, $\hat{Z}_{i_b} := Z_{i_a} - \beta_2$ for any $\beta_2 \in Z_{i_b} \setminus X_{i_b}$, and $\hat{Z} := Z + \beta_1 - \beta_2$. The constructed set $\hat{Z}$ is shown to be feasible in the following. Let $S \in \bar{S}_j$ be any scenario delaying T_{i_a} but not T_{i_b}, so $|\hat{Z} \cap X^S| - 1 = |Z \cap X^S|$, and assume, for contradiction, that $|\hat{Z} \cap X^S| = 2^k$. Consider $\hat{Z}$ for the scenario given by S but delaying T_{i_b} instead of T_{i_a}. The number of breaks of $\bar{Z}$ induced by S then reads as follows:

$$\begin{aligned} |\bar{Z} \cap X^S| &= |Z \cap X^S| + |Z_{i_b}| - |Z_{i_a}| \\ &\geq (|\hat{Z} \cap X^S| - 1) + (|X'_{i_b}| + 1) - (|X'_{i_a}| - 1) \\ &= 2^k - 1 + |X'_{i_b}| - |X'_{i_a}| + 2 \geq 2^k \end{aligned}$$

By this contradiction, the set $\hat{Z}$ is k-recoverable and furthermore $\sum_{t=1}^{j} |\hat{Z}_{i_t}| > \sum_{t=1}^{j} |Z_{i_t}|$. Repeating this construction eventually yields a maximum k-recoverable set $\hat{Z}$ with $\sum_{t=1}^{j} |\hat{Z}_{i_t}| = 2^k - 1$.

Now, let $a \in \{1, \ldots, j\}$ be an index such that $|Z_{i_a}| \leq |Z_{i_t}|$ for all $t \in \{1, \ldots, j\}$. Since $|X'| = 2^k - 1 = \sum_{t=1}^{j} |\hat{Z}_{i_t}|$ but $|Z| > |\bar{X}|$, there is an index $b \in \{j+1, \ldots, \ell\}$ with $|Z_{i_b}| > |X'_{i_b}|$. Consider the scenario $S \in \bar{S}_j$ delaying T_{i_b} as well as all the trains $T_{i_1}, \ldots, T_{i_j}$ except for T_{i_a}. As X'_{i_b} must have been decreased during the execution of Algorithm 3, the sets $|Z_{i_a}|$ and $|Z_{i_b}|$ satisfy $|Z_{i_b}| > |X'_{i_b}| = |X'_{i_j}| \geq |Z_{i_a}|$. This yields the following number of breaks of Z to remain unresolved under scenario S:

$$\sum_{t=1}^{j} |Z_{i_t}| - |Z_{i_a}| + |Z_{i_b}| > \sum_{t=1}^{j} |Z_{i_t}| = 2^k - 1$$

This contradicts Z being k-recoverable, so there exists no k-recoverable set of breaks that has greater size than $|\bar{X}|$, for which reason X' is optimal.

Analyzing the running time of Algorithm 3, the while-loop is iterated at most $n - 1$ times since $\sum_{t=1}^{j} |X_{i_t}| \leq n - 1$, and a single execution takes a time in $\mathcal{O}(\log \ell)$ if binary search is

used for updating α in line 5. Before the loop is entered, the sorting in line 1 takes $\mathcal{O}(\ell \log \ell)$ and setting α to its initial value $\mathcal{O}(\log \ell)$. This gives a total running time of $\mathcal{O}(n \log \ell)$, which is polynomial in the size of the input. □

As an immediate consequence of Theorem 11, the problem of recovery-robust train classification can be solved in polynomial time for the set of scenarios $\mathcal{S}_j$ by combining Algorithm 3 into Algorithm 2. The resulting algorithm is implemented in the following section and tested for a number of real-world classification instances.

4.4.4 Experimental Evaluation

The setup for testing the algorithm introduced in the previous section is presented first, with particular focus on the choice of the parameter values passed to the implementation, followed by the outcome of the evaluation.

Experimental Setup

For the evaluation of the robust algorithm for $\mathcal{S}_j$, the five real-world instances of Table 2.1 were used again. Note that these instances have been processed in Section 2.5 so that the cars have distinct types as required by the algorithm that is evaluated here. The algorithm was implemented in C++, compiled with the GNU compiler g++-4.4, and run on an 1.8 GHz Intel Core Duo CPU with 2 GB main memory.

Essentially, by adjusting the parameters p, k, and j, the algorithm allows flexibly trading off shortest schedules against the other extreme of strict robustness. Given some train classification instance, let $\underline{h}$ denote the length of an optimal non-robust schedule, which can be computed by Algorithm 1. Formally, this situation corresponds to $j = 0$ and a set of scenarios $\mathcal{S}_0$ containing only the empty scenario, i.e. the scenario inducing the original order of inbound trains, which is given by its set of induced breaks $X^S = \emptyset$. An optimal non-robust schedule of length $\underline{h}$ can also be computed with Algorithm 2 for input $\mathcal{S} = \mathcal{S}_0$, and, e.g., $k = p = 0$.

Moreover, let $\bar{h}$ denote the length of an optimal strictly robust schedule. This corresponds to a combination of parameter values $j \geq 1$ and $k = 0$ for any value of p. Algorithm 2 can thus

be run, e.g., with input $\mathcal{S} = \mathcal{S}_1$, $k = 0$, and $p = 0$ to compute a shortest strictly robust schedule. The maximum k-recoverable set of breaks returned by Algorithm 1 in line 9 of Algorithm 2 will then always be the empty set.

The values $\underline{h}$ and $\bar{h}$ present the lower and upper bounds for the length resulting from any combination of parameter values j, k, and p. Yet, as mentioned in Section 1.2.6, $\bar{h}$ may be exceeded by the geometric method, i.e. an optimal strictly robust schedule disregarding presorted inbound trains, and even longer schedules than this are obtained by triangular sorting. All the combinations of parameter values for p, k, and j are compared to each other and to triangular and geometric sorting in the following.

Experimental Results

Table 4.1 summarizes all the computed length values of optimal (p, k)-recovery-robust schedules according to the different choices of p, k, and j. Regarding the lower bounds on the length, `TUE` requires $\underline{h}= 3$, while all other instances yield $\underline{h}= 2$. With optimal non-robust schedules of five steps, the lower bound on the schedule length is given by $\bar{h} = 5$ for every instance. The geometric method requires $h = 5$ steps for `TUE` and `WED`, and even $h = 6$ for the other three instances. The triangular method needs eight steps for `TUE` and `WED`, nine for `WKD`, ten for `THU`, and even eleven steps for `MON`. These numbers show that ignoring presortedness of the inbound sequence of cars wastes a lot of potential for improvement.

For $k = 1$ corresponding to the smallest amount of recovery, allowing only one delayed train by putting $j = 1$ yields a schedule length that does not exceed $\underline{h}$ for the instances `MON` and `WKD` with $p = 0$, for `WED` and `THU` with $p \leq 1$, and for `TUE` even for $p \leq 3$. Hence, yet for lowest degrees of recovery, a certain level of robustness is obtained without increasing the schedule length beyond that of an optimal non-robust schedule. Incrementing the degree of disturbance to $j \geq 2$ without changing $k = 1$, a schedule length of $h = 4$ can still be obtained if the value of p is increased to $p = 1$ for `MON`, to $p = 2$ for `TUE`, `THU`, and `WKD`, and even to $p = 3$ for `WED`. These values are not only significantly smaller than those for the methods of geometric or triangular sorting. The length of $h = 4$ is also strictly better than the

	triangular	geometric	$k=$	0	0	1										
			$p=$	0	0	0		1		2		3		4		
			$j=$	1	0	1	2	1	2	1	2	1	2	1	2	
MON	11	6		5	2	2	4	3	4	3	5	4	5	4	5	...
TUE	8	5		5	3	3	4	3	4	3	4	3	5		5	
WED	8	5		5	2	2	4	2	4	3	4	3	4		5	
THU	10	6		5	2	2	4	2	4	3	4	3	5		5	
WKD	9	6		5	2	2	4	3	4	3	4	3	5		5	

	2														
	0		1				2				3			4	
	3	4	1	2	3	4	1	2	3	4	2	3	4	3	4
...	2	3	2	3	3	4	2	3	3	4	3	4	5	4	5
	3	3	3	3	3	3	3	3	3	4	3	4	4	4	4
	2	3	2	2	2	3	2	3	3	4	3	3	4		4
	2	3	2	2	2	3	2	3	3	4	3	3	4		5
	2	3	2	3	3	3	2	3	3	4	3	3	4		4

	3																4		
	0		1				2						3				1	2	
	7	8	5	6	7	8	3	4	5	6	7	8	4	5	6	7	8	1	13	14
...	2	3	2	3	3	3	2	3	3	3	3	4	3	3	3	4	4	2	2	2
	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	4	3	3	3
	2	3	2	2	2	3	2	2	2	3	3	3			3	3	4	2	2	2
	2	2	2	2	2	3	2	2	2	3	3	3			3	3	4	2	2	3
	2	2	2	2	3	3	2	2	3	3	3	3		3	3	3	4	2	2	2

Table 4.1: Optimal length values of (p, k)-recovery-robust schedules for S_j and the five traffic instances. Values for the triangular and geometric method are additionally given. Single omitted entries represent no meaningful choice of p. Completely omitted columns for smaller values of p or j do not differ from the columns corresponding to the smallest shown value. Similarly, if increasing j, p, or k does not change the length for any instance, the corresponding columns are omitted.

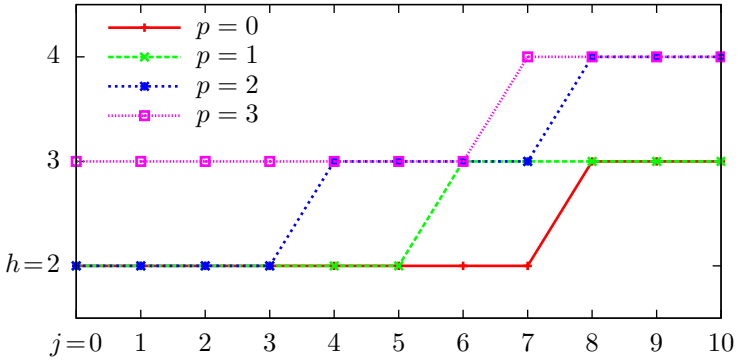


Figure 4.4: Optimal schedule length values h for the example instance MON and a choice of $k = 3$ for varying the value of p.

upper bound $\bar{h} = 5$, which indicates that strict robustness is rather wasteful for higher degrees of disruption.

The degree of robustness grows rapidly with increasing degrees of recovery. For $k = 4$ and $p \leq \underline{h}$, except for THU only with $p = 2$, any number of delayed trains can be allowed while still achieving the length $\underline{h}$ of an optimal non-robust schedule. Between these extremes, Table 4.1 shows how the relation of k and p affects the schedule length depending on the amount of delay j. For a fixed parameter value of $k = 2$ and arbitrarily high delays ($j \geq 4$), the value of p can be raised for $p = 1$ to obtain $h = 3$ for all instances except for MON. A schedule length of $h = 4$ can be achieved even for $p = 4$ for TUE, WED, and WKD. For $k = 3$, Figure 4.4 summarizes these values for the example of MON: a schedule of length $h = 3$ with a recovery starting after the third sorting step suffices to cope with a delay of up to six trains. This is just one step above the lower bound of $\underline{h} = 2$ of this instance. A value of $p = 1$ allows $h \leq 3$ even for any disturbance value j. These length values are all strictly smaller than the upper bound $\bar{h} = 5$ of the strictly robust schedules in spite of the moderate choices of recovery parameter values.

Similarly, when fixing the value of p, raising the value of k yields a rapid growth in robustness. For a fixed value of $p = 2$, Figure 4.5 shows how far the value of k must be increased to achieve a length of $\underline{h}$ for different amounts of delay. Except for

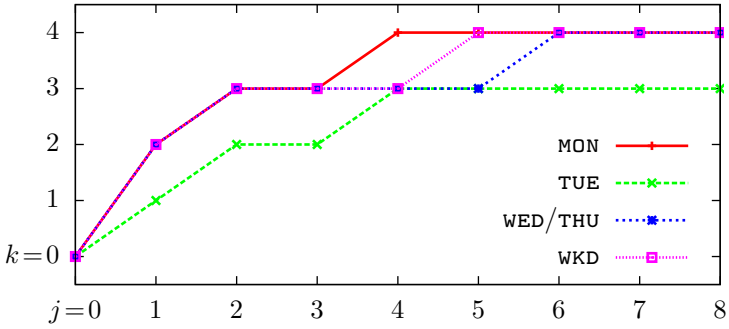


Figure 4.5: Lowest possible values of k to achieve the lower bound in schedule length for $p = 2$ for the five classification instances.

TUE, which is the only instance with $\underline{h}= 3$, the value of k must be raised rather quickly from $k = 0$ for $j = 0$ to $k = 3$ for $j = 2$ to achieve the lower bound length. Further increasing the degree of disruption eventually requires $k = 4$ for every instance but TUE. For higher disturbances of $j \geq 6$, however, the required value of k does increase anymore. A similar behavior can be observed for TUE, but generally on a lower level, with k growing from zero to two between $j = 0$ and $j = 2$, climbing to $k = 3$ for $j = 4$, and remaining on that level for any higher degrees of disruption.

The converse approach is plotted in Figure 4.6. The value of k is fixed by $k = 2$, and the maximum value of p is shown that allows a schedule length of $\underline{h}$. A small disturbance value of $j = 1$ allows $p = \underline{h}$ for all instances. For instance TUE this also holds for $j = 2$ and decreases to $p = 2$ for $j = 3$ and $p = 1$ for $j \in \{4, 5\}$. Independently of the choice of p, the lower bound length cannot be achieved for any higher values of j. The other instances show a more sudden drop in the possible value of p that still achieves $\underline{h}$. For $j = 2$ or $j = 3$, a value of $p \leq 1$ is required for WED and THU, and even $p = 0$ for MON and WKD. These four instances further have no feasible recovery-robust schedule for $j \geq 4$. Hence, higher values of k contribute much more to the power of the recovery action than low values of p.

Summarizing, by adjusting the recovery parameters k and p, the algorithm of Section 4.4.3 presents a flexible tool to trade off

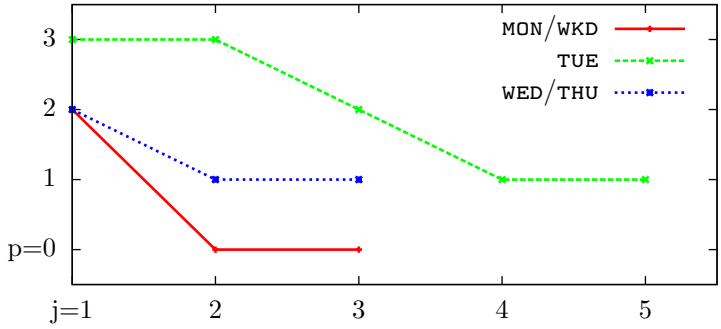


Figure 4.6: Smallest possible values of p to achieve a length of $\underline{h}$ depending on the amount of disruption for the different classification instances and a fixed value of $k = 2$.

between fast classification and robust schedules. Even for high degrees of robustness, much shorter classification schedules are achieved than by the strictly robust methods currently applied in practice.

4.5 Types of Cars

The considerations of this chapter so far assumed the cars of an instance to have distinct types. As mentioned in Section 4.1.2 above, this assumption is dropped and the generalization to types of cars is tackled in this section. The precise setting and relevant notation is given in Section 4.5.1, an optimal polynomial-time algorithm in Section 4.5.2, and an example in Section 4.5.3.

4.5.1 Model and Notation

The setting is mainly the one introduced in Section 4.1.2. As the only difference, the cars are not assumed to have distinct types here. The corresponding generalized order requirements were introduced in Sections 1.1.3 and 1.2.2. Recall the implicit indexing introduced in Section 1.2.2 for the setting with several cars of the same type: for any type $t \in \{1, \ldots, G\}$, the number of cars of this type in the inbound sequence $\mathcal{T} = \tau_1, \ldots, \tau_n$ of n cars is denoted by n^t; if $\tau_{i_1} = \tau_{i_2} = \ldots = \tau_{i_{n^t}} = t$ with

$i_1 < i_2 < \ldots < i_{n^t}$, are exactly these cars of type t, then t_1 refers to car τ_{i_1}, t_2 to car τ_{i_2}, etc. This notation will be used throughout this chapter. Referring to cars by their exact position in the inbound sequence of cars, as done in expression $\mathcal{T} = \tau_1, \ldots, \tau_n$ above, will be avoided in the remainder.

The polynomial-time algorithm for the class of scenarios $\mathcal{S}_j$ and distinct cars, which was dealt with in Section 4.4, is based on the fact that, for the dominating subset $\bar{\mathcal{S}}_j$, a potential break $(\tau, \tau{+}1)$ will necessarily be introduced if the inbound train containing τ is among the delayed trains. This does not hold for the setting with types of cars because the notion of potential breaks does not always apply in the form of Definition 4 here. The required generalized concept is given in the following definition.

Definition 11 *For any $t \in \{1, \ldots, G-1\}$, the pair of consecutive types $(t, t+1)$ is called an* original break *if t_1 occurs in the inbound sequence of cars at a later position than $(t{+}1)_{n^{t+1}}$. Otherwise, $(t, t{+}1)$ is called a* potential break.

Put differently, a pair $(t, t+1)$ of consecutive types is *not* a potential break if all cars of type $t{+}1$ arrive before the first car t_1 of the preceding type t arrives. Note that, if every type occurs only once, Definition 11 degenerates to the notions of original and potential breaks of Definition 4.

The complexity status of the full generalization to types of cars remains open. Here, a special case is presented where each single car represents its own train, which will be called a *singleton train*. A dynamic programming approach is followed to derive an algorithm for calculating optimal recovery robust classification schedules for $\mathcal{S}_j$ in time polynomial in the number of cars.

In the setting with singleton trains, delaying one train, i.e. one car, results in at most one unresolved break, which will be made use of here. If $j \leq 2^k - 1$, the optimal non-robust schedule is in fact recovery robust since the number of unresolved breaks induced in any block is always bounded by j for singleton trains. For any sensible choice of $p \leq h$, if $G \leq 2^p$, no recovery is needed at all since assigning every car of type t to the binary representation $[t]_2$ of t yields a strictly robust schedule of length at most h. Therefore, $G > 2^p$ and particularly $j > 2^k - 1$ is assumed in the following.

4.5.2 Optimal Algorithm

Recall the order relation (1.4) defined in Chapter 1: $t_i > s_j$ iff either $t > s$ or $t = s$ and $i < j$. This yields the ascending order of cars $1_{n^1}, \ldots, 1_1, 2_{n^2}, \ldots, 2_1, \ \ldots \ , T_{n^G}, \ldots, T_1$. Also, as shown in Lemma 2 above, the sequence of bitstrings w.r.t. this order can be assumed to be non-decreasing in any feasible schedule.

In the algorithm blocks will be grown one after the other similar to Section 4.3.1. For each block a two-dimensional table will be built containing up to n columns corresponding to the cars not yet assigned to bitstrings in previous blocks. The columns are indexed from left to right by the cars in ascending order w.r.t. ordering (1.4). There are (up to) $2^p + 1$ rows indexed $d = 0, \ldots, 2^p$. A table entry (d, t_i) corresponds to a feasible partial first-stage schedule for the cars in the block ending here with t_i as its greatest contained car (w.r.t. (1.4)); the entry's value $c(d, t_i)$ equals the optimum number of potential breaks that may be induced in this block if car t_i is assigned to the dth bitstring as well as all cars t_j with $j > i$.

Recalling the Order of Types

When deriving the dynamic programming tables as described in the following paragraphs, a partial function $\mathrm{succ}(t_i)$ for types $t \in \{2, \ldots, G\}$ and indices $i \in \{1, \ldots, n^t\}$ will be used that maps each car t_i to the index of the first car of type $t - 1$ arriving after t_i if such a car exists. This will particularly be used for checking whether a car t_i arrives earlier than all cars of type $t-1$ by verifying $\mathrm{succ}(t_i) = 1$. $\mathrm{succ}(t_i)$ is not defined for cars t_i arriving later than all cars of type $t - 1$.

The partial function can be implemented as an array of arrays with constant time random access. It is built in time $\mathcal{O}(G + n) = \mathcal{O}(n)$ using G buckets corresponding to the cars' types: the inbound sequence of cars is scanned in the order of arrival from the first to the last car; when car t_i is considered, i is put in the tth bucket, all cars currently in the $(t + 1)$th bucket are mapped to i (unless $t = G$), and the $(t+1)$th bucket is emptied. The data structure is defined to yield $\mathrm{succ}(t_i) = \mathrm{nil}$ for the undefined entries mentioned above, so the values range in $\{\mathrm{nil}, 1, 2, \ldots, \max_{1 \le i \le G-1} n^i\}$.

Initialization

Let s_α denote the smallest car (w.r.t. (1.4)) that has not been considered yet in any iteration corresponding to any previous block. All entries $(0, t_i)$, $t_i \geq s_\alpha$, of the first row are put $c(0, t_i) = \infty$, which means that there exists no (partial) solution for a non-zero number of cars using no bitstrings. (This is also done in the dynamic programming table 4.2 for the example in Section 4.5.3.) The entries (d, t_α), $d = 1, \ldots, 2^p$, of the column corresponding to the smallest unassigned car are initialized with zero, which means assigning the dth bitstring of this block to all cars $t_\alpha, \ldots, t_{n^t}$ of type t yields the optimal number of zero unresolved breaks. Each entry (d, t_i), $d = 1, \ldots, 2^p$, $i = \alpha+1, \ldots, n^t$ of any car of type t arriving later than t_α does not correspond to a sensible schedule as cars $t_\alpha, \ldots, t_{i-1}$ would remain unassigned by the definition of a table entry further above, so these entries are put $c(d, t_i) = \infty$ in the initialization.

Entry Derivation

There are two main derivation rules for table entries, the first of which applies to entries (d, t_1) corresponding to cars that are the first of their type to arrive. the dth bitstring may be assigned to no car of type $t-1$ in order to resolve the potential break $(t-1, t)$ in the first-stage schedule. This means all cars of type $t-1$ are assigned to the $(d-1)$th bitstring, so $c(d, t_1) = c(d-1, (t-1)_1)$. Note that this rule also applies in case $(t-1, t)$ is an original break, i.e., $\mathrm{succ}(t_{n^t}) = 1$ and all cars of type t arrive before $(t-1)_1$. In this case, the rule must be applied for the first-stage solution to be a feasible schedule. However, if at least one car of type $t-1$ arrives before t_1, i.e., if $\mathrm{succ}(t_1) > 1$, the dth bitstring may additionally be assigned greedily to all those cars of type $t-1$ arriving before t_1. This leaves the potential break $(t-1, t)$ unresolved and thus yields $c(d, t_1) = c(d-1, (t-1)_{\mathrm{succ}(t_1)}) + 1$. Together, the first rule reads $c(d, t_1) = \min\big(c(d-1, (t-1)_1), c(d, (t-1)_{\mathrm{succ}(t_1)}) + 1\big)$. As a special case of this rule, $\mathrm{succ}(t_1) = \mathrm{nil}$ will hold if all cars of type $t-1$ arrive before t_1. Then, $c(d, (t-1)_1)$ must be considered instead of $c(d, (t-1)_{\mathrm{succ}(t_1)})$ since the dth bitstring is assigned to *all* cars of type $t-1$.

The second rule applies to entries (d, t_i) with $i > 1$, cor-

responding to cars t_i which are not the first of their types to arrive. If at least one car of type $t-1$ arrives before t_i, which means $\mathrm{succ}(t_1) > 1$, the dth bitstring can be greedily assigned to all cars of type $t-1$ arriving before t_i as well as to the cars $t_i, t_{i+1}, \ldots, t_{n^t}$. This results in the potential break $(t-1, t)$ to remain unresolved, so $c(d, t_i) = c(d-1, (t-1)_{\mathrm{succ}(t_i)})+1$. Again, for the special case $\mathrm{succ}(t_i) = \mathrm{nil}$, which means all cars of type $t-1$ arrive before t_i, $c(d, t_i) = c(d, (t-1)_1) + 1$ since all cars of type $t-1$ are assigned to the d then. Note that $\mathrm{succ}(t_i) = 1$ corresponds to all cars of type $t-1$ arriving *after* t_i. In this case the entry (d, t_i) does not correspond to a complete schedule since the cars $t_1, \ldots, t_{i-1}$ would remain unassigned. In the example of Section 4.5.3, this is the case for the entries of the column corresponding to 6_3.

Summarizing, the following four cases are distinguished when deriving the table entries, of which the first two correspond to the first and the third and fourth to the second rule from above:

$$c(d, t_i) = \begin{cases} \min\left(c(d\text{–}1, (t\text{–}1)_1), c(d, (t\text{–}1)_{\mathrm{succ}(t_1)}) + 1\right), \\ \qquad\qquad \text{if } i = 1 \text{ and } \mathrm{succ}(t_1) \neq \mathrm{nil} \\ \min\left(c(d\text{–}1, (t\text{–}1)_1), c(d, (t\text{–}1)_1) + 1\right), \\ \qquad\qquad \text{if } i = 1 \text{ and } \mathrm{succ}(t_1) = \mathrm{nil} \\ c(d\text{–}1, (t\text{–}1)_{\mathrm{succ}(t_i)}) + 1, \quad \text{if } i > 1 \text{ and } \mathrm{succ}(t_i) \neq \mathrm{nil} \\ c(d\text{–}1, (t\text{–}1)_1) + 1, \qquad \text{if } i > 1 \text{ and } \mathrm{succ}(t_i) = \mathrm{nil} \end{cases}$$

Each table can be filled row-wise or column-wise as entries are only used from the rectangular area with the entry to be calculated in the bottom-right corner.

End of Block and Backtracking

Finally, in the row corresponding to $d = 2^p$, the "right-most" entry (d, t_i), meaning the entry that maximizes t_i w.r.t. (1.4), for which $c(d, t_i) \leq 2^k - 1$ is still satisfied determines the end of this block. The car t_i of this entry then presents the greatest car w.r.t. (1.4) contained in the block. For the last of all blocks, it may happen that increasing d does not result in any improvement from some point. If a line for some value of d looks exactly like that for $d-1$, the algorithm can be cancelled. Note that the value $c(2^p, t_i)$ can be much smaller than $2^k - 1$ due to

a relatively small value of p in combination with high numbers of original breaks.

The partial schedule for this block, i.e. the bitstring assignment of its cars, is reconstructed as follows, starting with the entry (d, t_i) corresponding to the end of the block. if $i = 1$, i.e., if $c(d, t_i)$ has been computed by the first rule, cars $t_1, \ldots, t_{n^t}$ are assigned to the dth bitstring. If the first part of the first rule has been applied, the backtracking just proceeds with $(d, (t-1)_1)$. But, if the greedy second part of the first rule has been applied, also the cars $(t-1)_1, \ldots, (t-1)_{\mathrm{succ}(t_i)-1}$ are assigned to d, and entry $(d-1, (t-1)_{\mathrm{succ}(t_1)})$ is considered in the next step. For the special case part of this rule, assigning the cars of type $t-1$ is postponed to the next backtracking step, which is carried out for $(d, (t-1)_1)$.

For $i > 1$, corresponding to the second rule, the dth bitstring is assigned to $t_i, \ldots, t_{n^t}$ as well as $(t-1)_1, \ldots, (t-1)_{\mathrm{succ}(t_1)-1}$, and $(d-1, (t-1)_{\mathrm{succ}(t_i)})$ is considered in the next step. In the special case $\mathrm{succ}(t_i) = \mathrm{nil}$ corresponding to the fourth case, the dth bitstring is only assigned to $t_i, \ldots, t_{n^t}$ in the current step, and the next step considers entry $(d, (t-1)_1)$.

The backtracking for a partial classification schedule finishes when the first row of the table corresponding to $d = 0$ is reached. Then, the reconstruction continues with the table of the previous block.

Running Time

The table of any single block has at most $2^p + 1$ rows and n columns and is initialized in $\mathcal{O}(n + 2^p) = \mathcal{O}(n)$, which yields $\mathcal{O}(n^2)$ for initializing all tables. Every entry $c(d, t_i)$ can be derived in time $\mathcal{O}(1)$ for either rule. Hence, calculating a complete row takes time $\mathcal{O}(n)$, which means $\mathcal{O}(n^2)$ for deriving all entries of all tables since at most n bitstrings will ever be used. The complete backtracking takes time linear in the schedule length $\mathcal{O}(h) \subseteq \mathcal{O}(n)$, while explicitly writing out the bitstrings of the schedule takes $\mathcal{O}(hn) \subseteq \mathcal{O}(n^2)$. As mentioned above, building the succ-data structure can be accomplished in time $\mathcal{O}(n)$, which does not affect the total running time of $\mathcal{O}(n^2)$.

The deduced algorithm is summarized in the following theorem as the main result of this section. An example is given further below.

d	1_2	1_1	2_2	2_1	3_2	3_1	4_2	4_1	5_2	5_1	6_3	6_2	6_1
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	∞	**0**	1	∞	∞	∞	∞	∞	∞	∞	$\times$	∞	∞
2	∞	0	1	**0**	**1**	∞	∞	∞	∞	∞	$\times$	∞	∞
3	∞	0	1	0	1	0	2	**2**	3	∞	$\times$	∞	∞
4	∞	0	1	0	1	0	2	0	1	**2**	$\times$	4	∞

Table 4.2: The table for the first block in the dynamic programming approach of the example of this section. The entries corresponding to the optimal solution are highlighted.

Theorem 12 *Let $j, p, k \geq 0$ be nonnegative integers. Given an expected inbound sequence of n singleton trains $T_1, \ldots, T_n$ with non-distinct types, a (p, k)-recovery robust classification schedule of minimum length for the set of scenarios $\mathcal{S}_j$ can be computed in polynomial time $\mathcal{O}\left(n^2\right)$.*

4.5.3 Example

An example run of the algorithm presented in the previous section is given here, comprising the derivation of the dynamic programming table for the first block and the corresponding reconstruction of the computed schedule. Put $k = 2$ and $p = 2$, and consider the following inbound sequence of cars/trains:

$$7_1, 6_1, 5_1, 6_2, 6_3, 3_1, 7_2, 8_1, 4_1, 1_1, 4_2, 2_1, 5_2, 1_2, 2_2, 3_2$$

The number of cars in this sequence is $n = 16$ and the number of types is $G = 7$.

Table 4.2 shows the derivation corresponding to the first block of the optimal schedule. As explained above, the first row is initialized with ∞ and the other entries of the column of car 1_1 with zero. The entries of 1_2 do not correspond to complete schedules as car 1_1 would remain unassigned by the definition of a table entry. (A similar statement holds for 6_3.) Finally, entry $(4, 5_1)$ is the right-most entry (d, t_i) with $d = 2^p$ and $c(d, t_i) \leq 2^k - 1$. Note that entries (d, t_i) with $t > 6$ need no consideration since there is no entry $c(d, 6_i) \leq 2^k - 1$.

In the schedule reconstruction, the four bitstrings $[0]_2 = 00$, $[1]_2 = 01$, $[2]_2 = 10$, and $[3]_2 = 11$ of length two are used.

When the backtracking starts at entry $(4, 5_1)$, the fourth bitstring $[3]_2 = 11$ is assigned to 5_1 and also to 5_2. Entry $(4, 5_1)$ has been derived from $(3, 4_1)$ by the first part of the general case of the first rule, so the potential break $(3, 4)$ is resolved by mapping both cars 4_1 and 4_2 of type 4 to the smaller bitstring $[2]_2 = 10$. $(3, 4_1)$ has been computed from entry $(2, 3_2)$ by the greedy second part of the general case of the first rule, which means that the same bitstring 10 is also assigned car 3_1, whereas 3_2 is assigned to $[1]_2 = 01$. This entry $(2, 3_2)$ comes from $(2, 2_1)$ using the special case of the second rule, which means that also all the cars of type 2 are assigned to the same bitstring as 3_2. Finally, entry $(1, 1_1)$ has been the origin of deriving $(2, 2_1)$ with the first part of the general case of the first rule, so all cars of type 1 get the smallest available bitstring $[0]_2 = 00$.

In the resulting partial schedule of the first block, the two potential breaks $(2, 3)$ and $(3, 4)$ thus remain unresolved, the former by having applied the second rule, the latter by the second part of the first rule. They will be induced at the same time by a scenario that, e.g., includes delaying 2_2 and 3_1. Note that $j > 2^k - 1$ is assumed. The dynamic programming table for the second block then considers all the unassigned cars $t_i > 5_1$, which are all the cars of type 6 in this case.

4.6 Summary

A polynomial-time algorithm for deriving recovery-robust train classification schedules of minimum length has been derived in this chapter. In contrast to [CDD+07, CDD+09], this approach includes multiple inbound trains. This allowed considering the most relevant disturbance in practice, which is the delay of trains. For the natural recovery action of (p, k)-extensions applied in the setting of general scenarios, the problem turned out to be $\mathcal{NP}$-hard for every constant $k \geq 1$.

Nevertheless, for the simple yet general set of scenarios S_j, the generic algorithm developed in Section 4.3 can be implemented in polynomial time since in this case the subproblem of calculating a maximum recoverable set of breaks can be solved efficiently. The experimental study of Section 4.4.4 finally indicates that the resulting algorithm improves on the current classification practice as it yields shorter schedules while still allowing

high degrees of robustness. Its general flexibility of balancing between strictly robust and optimal non-robust schedules raises potential for increased traffic throughput in classification yards.

For the general setting of non-distinct cars, i.e. cars with types, a special case with singleton trains was considered in Section 4.5. With a generalized definition of the concept of break, a dynamic programming approach was applied to find (p, k)-recovery-robust classification schedules of minimum length for the class of scenarios $\mathcal{S}_j$ in polynomial time also in this setting.

Chapter Notes The results of Sections 4.2 to 4.4 were originally published in [BM10], which is a joint work with Christina Büsing, to which both authors contributed equal amounts. The same results are also contained in [Büs10]. The result of Section 4.2.2 above, i.e. the fact that inserting right after the offset p always presents an optimal way of inserting, is only mentioned in [BM10] without the formal proofs given here.

The problem of Section 4.5 was reduced in [Büs10] to a resource-constraint shortest path problem, which presents an alternative way of solving this problem in polynomial time. Beyond the result of Section 4.5, the approach of [Büs10] works in polynomial time not only for the class of scenarios $\mathcal{S}_j$, but also for any set of scenarios that has constant size. The dynamic programming approach of Section 4.5, which directly exploits the structure of $\mathcal{S}_j$, has not been previously published.

Future Directions Further practical restrictions, such as the limited number of classification tracks considered in Section 3.2, are desirable to be integrated in the context of robustness. Also, as mentioned in Section 1.1.4, the number of cars rolled in, i.e. the weight of the schedule, yields a secondary objective that can be additionally minimized for an optimal length. Most importantly, it remains an open question whether the case of general, non-singleton trains with non-distinct cars can be solved efficiently for $\mathcal{S}_j$ or other natural sets of scenarios with more than constant size.

Conclusion

In this thesis the algorithmic foundations of multistage train classification have been studied. Section 1.1 has shown how rich in variations the underlying practical railway problem is. Developing a theoretical model for all these aspects in Section 1.2, including different objectives, infrastructural variants, and operational constraints, has led to an abstract problem yielding a wide range of research questions, which makes the problem an interesting field of theoretical computer science research.

As an important part of the abstract model, the novel encoding of classification schedules of Section 1.2 has enabled an efficient representation of existing schedules, from which important characteristics of the represented sorting process could directly be read off. This has not only allowed a precise analysis of the commonly used methods in practice as presented in Section 1.2.6, it has also been successfully applied to derive multistage classification schedules for the most basic problem setting that provably minimize the number of required sorting steps. Finally, with the formal concepts of breaks and chains, optimizing the order of inbound trains has been shown to be an $\mathcal{NP}$-hard problem.

With the strong theoretical model established in Chapter 1, algorithmic solutions for several variants of the problem, resulting from different practical aspects, have been developed in the successive chapters. For the $\mathcal{NP}$-hard train classification problem with limited track capacities, Chapter 2 introduces an integer programming model as well as several approximation algorithms. The thorough experimental evaluation of the approaches has yielded valuable insights about their practical behavior, which provides approaches for improving the real-world classification methods applied in practice today.

Similarly, for several other practical constraints in Chapter 3, the limitations in performance have been explored and powerful solution approaches have been provided. Starting from considerations for the sole constraints, an integrated integer programming approach has been developed, showing the high flexibility of the developed theoretical model. The application to real-world traffic data has indeed brought forth a solution that outperforms the originally applied schedule. The shorter sorting process with less engine movement, reduced wear of switches, and more efficient use of track space proves the applicability in practice and thus the practical significance of the theoretical considerations made.

Finally, the fact that every railway network is subject to disruptions motivates considering robustness aspects of train classification, which has been done in Chapter 4. For the notion of recoverable robustness, the problem of finding optimal recovery-robust solutions has turned out to be $\mathcal{NP}$-hard for allowing any amounts of delayed trains as the general scenario for disturbance. However, a restricted but still quite general model of delays could be developed for which an optimal polynomial-time algorithm has been derived. Again, the experimental study of that chapter has indicated that implementing these solutions may improve on the current practice. Particularly, this method may introduce more flexibility to balance between the conflicting objectives of having robust but also short classification procedures.

All in all, this thesis has provided a deeper understanding of the inherent combinatorial structure in the practical train classification problem. While being an appealing subject of theoretical research on its own, the insights gained from the abstract problem have also resulted in various efficient algorithmic solutions that can actually be applied in practice. This provides a fundamental step towards improved train classification in practice, with more efficiency and a raised potential for increased traffic throughput, as a contribution to promote railway freight transport in general.

Bibliography

[AKK07] Norbert Adamko, Antonín Kavička, and Valent Klima. Villon - Agent based generic simulation model of transportation logistic terminals. In *Proc. of the 2007 European Simulation and Modelling Conference (ESM-07)*, pages 364–368, 2007.

[Bau59] Oskar Baumann. Die Planung der Simultanformation von Nahgüterzügen für den Rangierbahnhof Zürich-Limmattal. *Eisenbahntechnische Rundschau*, 19, Sonderausgabe 11 Rangiertechnik:25–35, 1959.

[BFMM11] Markus Bohlin, Holger Flier, Jens Maue, and Matúš Mihalák. Hump yard track allocation with temporary car storage. In *4th International Seminar on Railway Operations Modelling and Analysis (RailRome2011)*. International Association of Railway Operations Research (IAROR), 2011.

[BGSR80] Lawrence D. Bodin, Bruce L. Golden, Allan D. Schuster, and William Romig. A model for the blocking of trains. *Transportation Research Part B*, 14(1–2):115–120, 1980.

[BM10] Christina Büsing and Jens Maue. Robust algorithms for sorting railway cars. In *Proc. of the 18th Annual European Symposium on Algorithms (ESA-10)*, volume 6346 of *LNCS*, pages 350–361. Springer, 2010.

[Bón02] Miklós Bóna. A survey of stack-sorting disciplines. *Electronic Journal of Combinatorics*, 9(2), 2002.

[Boo57] B. C. M. Boot. Zugbildung in Holland. *Eisenbahntechnische Rundschau*, 17, Sonderausgabe 8 Rangiertechnik:28–32, 1957.

[BS04] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[Büs10] Christina Büsing. *Recoverable Robustness in Combinatorial Optimization.* PhD thesis, Technische Universität Berlin, December 2010.

[CCG+08] Valentina Cacchiani, Alberto Caprara, Laura Galli, Leo Kroon, and Gábor Mároti. Recoverable robustness for railway rolling stock planning. In *Proc. of the 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS-08)*, volume 9 of *OASIcs*. LZI Schloss Dagstuhl, Wadern, Germany, 2008.

[CDD+07] Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. Robust algorithms and price of robustness in shunting problems. In *Proc. of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS-07)*, volume 7 of *OASIcs*, pages 175–190. LZI Schloss Dagstuhl, Wadern, Germany, 2007.

[CDD+08] Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. Delay management problem: Complexity results and robust algorithms. In *Proc. of the 2nd Annual Conference on Combinatorial Optimization and Applications (COCOA-08)*, volume 5165 of *LNCS*, pages 458–468. Springer, 2008.

[CDD+09] Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, Alfredo Navarra, Michael Schachtebeck, and Anita

Schöbel. Recoverable robustness in shunting and timetabling. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 28–60. Springer, 2009.

[CDSS08] Serafino Cicerone, Gabriele Di Stefano, Michael Schachtebeck, and Anita Schöbel. Dynamic algorithms for recoverable robustness problems. In *Proc. of the 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS-08)*, volume 9 of *OASIcs*. LZI Schloss Dagstuhl, Wadern, Germany, 2008.

[CTV98] Jean-Francois Cordeau, Paolo Toth, and Daniele Vigo. A survey of optimization models for train routing and scheduling. *Transportation Science*, 32(4):380–404, 1998.

[Dag86] Carlos F. Daganzo. Static blocking at railyards: Sorting implications and track requirements. *Transportation Science*, 20(3):189–199, 1986.

[Dag87a] Carlos F. Daganzo. Dynamic blocking for railyards: Part I. homogeneous traffic. *Transportation Research*, 21B(1):1–27, 1987.

[Dag87b] Carlos F. Daganzo. Dynamic blocking for railyards: Part II. heterogeneous traffic. *Transportation Research*, 21B(1):29–40, 1987.

[Dah08] Elias Dahlhaus. How to get as many cars sorted as possible in a hump yard using only one sorting step. In *17th International Symposium EURNEX-Žel*, 2008.

[DDH83] Carlos F. Daganzo, Richard G. Dowling, and Randolph W. Hall. Railroad classification yard throughput: The case of multistage triangular sorting. *Transportation Research*, 17A(2):95–106, 1983.

[DHMR00] Elias Dahlhaus, Peter Horák, Mirka Miller, and Joseph F. Ryan. The train marshalling problem. *Discrete Applied Mathematics*, 103(1–3):41–54, 2000.

[DMMR00] Elias Dahlhaus, Fredrik Manne, Mirka Miller, and Joe Ryan. Algorithms for combinatorial problems related to train marshalling. In *Proc. of the 11th Australasian Workshop on Combinatorial Algorithms (AWOCA-00)*, pages 7–16, 2000.

[DSMM+07] Gabriele Di Stefano, Jens Maue, Maciej Modelski, Alfredo Navarra, Marc Nunkesser, and John van den Broek. Models for rearranging train cars. Technical Report TR-0089, ARRIVAL, 2007.

[EKMZ04] Michael Edinger, Rainer König, Peter Márton, and Miloš Zatko. Die rechnergestützte Simulation des Betriebs in Werkbahn BASF Ludwigshafen. In *Railways on the Edge of the 3rd Millennium (ZEL-04)*, pages 161–165, 2004.

[End63] Karl Endmann. Untersuchungen über die Simultanzugbildung bei der Deutschen Bundesbahn. *Bundesbahn*, 37:593–600, 1963.

[FG10] Holger Flier and Michael Gatto. personal communication, 2010.

[Fla53] Hermann Flandorffer. Vereinfachte Güterzugbildung. *Eisenbahntechnische Rundschau*, 13, Sonderausgabe 2 Rangiertechnik:114–118, 1953.

[FM09] Matteo Fischetti and Michele Monaci. Light robustness. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 61–84. Springer, 2009.

[FPR09] Paola Festa, Panos M. Pardalos, and Mauricio G. C. Resende. Feedback set problems. In *Encyclopedia of Optimization*, pages 1005–1016. Springer, 2nd edition, 2009.

[GJ79a] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. problem GT20.

[GJ79b] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. problem LO6.

[GMMW09] Michael Gatto, Jens Maue, Matúš Mihalák, and Peter Widmayer. Shunting for dummies: An introductory algorithmic survey. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 310–337. Springer, 2009.

[Har00] E. Hunter Harrison. How CN runs a scheduled railroad. *Railway Age*, December 2000.

[HBSM95] Christopher L. Huntley, Donald E. Brown, David E. Sappington, and Bernard P. Markowicz. Freight routing and scheduling at CSX Transportation. *Interfaces*, 25(3):58–71, 1995.

[HL10] Thomas Heydenreich and Mathias Lahrmann. How to save wagonload freight. *Railway Gazette International*, 166(9):126–129, September 2010.

[HM10] Alain Hauser and Jens Maue. Experimental evaluation of approximation and heuristic algorithms for sorting railway cars. In *Proc. of the 9th International Symposium on Experimental Algorithms (SEA-10)*, volume 6049 of *LNCS*, pages 154–165. Springer, 2010.

[Hol07] Hans Peter Holliger. Rangierbahnhof Limmattal. personal communication, 2007.

[HZ07] Ronny S. Hansmann and Uwe T. Zimmermann. Optimal sorting of rolling stock at hump yards. In *Mathematics - Key Technology for the Future: Joint Projects Between Universities and Industry*, pages 189–203. Springer, 2007.

[JAŞ08] Krishna C. Jha, Ravindra K. Ahuja, and Güvenc Şahin. New approaches for solving the block-to-train assignment problem. *NETWORKS*, 51:48–62, 2008.

[JMMN07] Riko Jacob, Peter Márton, Jens Maue, and Marc Nunkesser. Multistage methods for freight train classification. In *Proc. of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS-07)*, volume 7 of *OASIcs*, pages 158–174. LZI Schloss Dagstuhl, Wadern, Germany, 2007.

[JMMN11] Riko Jacob, Peter Márton, Jens Maue, and Marc Nunkesser. Multistage methods for freight train classification. *Networks*, 57(1):87–105, 2011.

[Kec58] Willi Keckeisen. Bau und Betrieb der Stuttgarter Hafenbahn. *Eisenbahntechnische Rundschau*, 7(10):408–420, 1958.

[KL08] Felix G. König and Marco E. Lübbecke. Sorting with complete networks of stacks. In *Proc. of the 19th International Symposium on Algorithms and Computation (ISAAC-09)*, volume 5878 of *LNCS*, pages 895–906. Springer, 2008.

[Knu97] Donald E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, 3rd edition, 1997.

[Kre62] Karl Krell. Grundgedanken des Simultanverfahrens. *Eisenbahntechnische Rundschau*, 22, Sonderausgabe 15 Rangiertechnik:15–23, 1962.

[Kre63] Karl Krell. Ein Beitrag zur gemeinsamen Nutzung von Nahgüterzügen. *Eisenbahntechnische Rundschau*, 23, Sonderausgabe 16 Rangiertechnik:16–25, 1963.

[Kum04] Sadhir Kumar. *Improvement of Railroad Yard Operations*, chapter 25, pages 25.1–25.28. McGraw-Hill, 2004.

[LLMS07] Christian Liebchen, Marco Lübbecke, Rolf H. Möhring, and Sebastian Stiller. Recoverable robustness. Technical Report TR-0066, ARRIVAL, 2007.

[LLMS09] Christian Liebchen, Marco E. Lübbecke, Rolf H. Möhring, and Sebastian Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 1–27. Springer, 2009.

[Már05] Peter Márton. Experimental evaluation of selected methods for multigroup trains formation. *Communications*, 2:5–8, 2005.

[MMN09] Peter Márton, Jens Maue, and Marc Nunkesser. An improved classification procedure for the hump yard Lausanne Triage. In *Proc. of the 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS-09)*, volume 12 of *OASIcs*. LZI Schloss Dagstuhl, Wadern, Germany, 2009.

[MN09] Jens Maue and Marc Nunkesser. Evaluation of computational methods for freight train classification schedules. Technical Report TR-0184, ARRIVAL, 2009.

[Pen59] K. J. Pentinga. Teaching simultaneous marshalling. *The Railway Gazette*, pages 590–593, 1959.

[Pet77a] E. R. Petersen. Railyard modeling: Part I. Prediction of put-through time. *Transportation Science*, 11(2):37–49, 1977.

[Pet77b] E. R. Petersen. Railyard modeling: Part II. The effect of yard facilities on congestion. *Transportation Science*, 11(2):50–58, 1977.

[Sid72] M. W. Siddiqee. Investigation of sorting and train formation schemes for a railroad hump yard. In *Proc. of the 5th International Symposium on the*

Theory of Traffic Flow and Transportation, pages 377–387, 1972.

[Tar72] Robert Endre Tarjan. Sorting using networks of queues and stacks. *Journal of the ACM*, 19(2):341–346, 1972.

[Vil] Simcon s.r.o., Žilina, Slovakia. *Introduction to Villon*.

[ZL06] Miloš Zatko and Stephan Leber. Simulation komplexer Betriebsprozesse in einem Rangierbahnhof am Beispiel von Lausanne Triage. *Schweizer Eisenbahn-Revue*, 11:9500–9503, 2006.

Nomenclature

Classification Yard

C	capacity of sorting track
W	number of sorting tracks

In- and Outbound Trains

ℓ	number of inbound trains
n'_k	length of the kth inbound train
T_k	kth inbound train
$\mathcal{T}$	inbound sequence of cars resulting from concatenation of inbound trains
m	number of outbound trains
n_k	length of the kth outbound train
g_k	number of groups in the kth outbound train
U	outbound train (for special case of a single outbound train)
U_k	kth outbound train (for general case of multiple outbound trains)

Cars

n	total number of cars
τ, σ	a car (in case of distinct cars)
τ_x	xth car of the inbound sequence of cars
$\tau_{k,i}$	ith car of the kth inbound train (in explanations of Section 1.2.2)

Types

t	a type
n^t	number of cars of type t
G	maximum type
t_i, s_k	ith car of type t, kth car of type s in the inbound sequence of cars, resp.
$t_{(\alpha)}$	last car of type t in the inbound sequence of cars
G^j	greatest occurring type of the jth outbound train

Classification Schedules

$[k]_2$	binary representation of an integer k
b, b'	a bitstring
B	a classification schedule
h	length of some schedule
$w(B)$	weight of schedule B
b_i	ith column of schedule B
$w(b_i)$	weight of column b_i
b^τ, b^{τ_x}	bitstring assigned to car τ, τ_x, resp., in schedule B
b_i^τ, $b_i^{\tau_x}$	ith bit of bitstring b^τ, b^{τ_x}, resp.
B^*	an optimal classification schedule, i.e. a feasible schedule of minimum length
h^*	length of an optimal schedule
$\tilde{B}$	an intermediate schedule, i.e. an optimal schedule satisfying (2.4)
$\tilde{h}$	length of an intermediate schedule
$\tilde{B}_k$	partial intermediate schedule of kth outbound train (Section 2.4)
B'_k	partial intermediate schedule without leading zero-columns (Section 2.4.3)
h'	actual length of some partial intermediate schedule
$\bar{B}_k$	intermediate schedule under construction after kth concatenation (Section 2.4.3)

Scenarios and Sets of Breaks

k	number of columns to be inserted
p	offset for inserting columns
X_{org}	set of original breaks
$X_{(\tau,\sigma)}$	set of all original and potential breaks between cars τ and σ
$\mathcal{S}$	a set of scenarios
S	a scenario
X^S	subset of potential breaks induced by scenario S
Y_x	set of unresolved breaks in xth block of some schedule

Shunting Glossary

classification bowl subyard of a classification yard containing the wide set of long, parallel classification tracks; connected by a tree of switches to the lead track at one end and to a departure yard or directly to line tracks at the other.

classification track track of the classification bowl used to separate cars of an inbound train, temporarily store cars during a classification process, or form outbound trains.

classification yard installation of railway tracks and switches that serves the purpose of separating freight cars to different tracks and making them into new trains.

departure yard set of parallel tracks in a classification yard, situated between the classification bowl and the line track, to which completed trains are hauled before they depart.

flat yard classification yard without a hump, in which cars are pushed by shunting engines to sort them; contrasts to hump yard.

hump yard classification yard with a hump at the end of the lead track towards the bowl, over which the cars are pushed by a shunting engine and roll by gravity to the classification tracks; contrasts to flat yard.

lead track track on which cars are decoupled and prepared to be pushed to the classification bowl, to which the lead track is connected by a tree of switches.

local freight train freight train serving multiple destinations along its route and dropping a number of freight cars at each station.

output track classification track at the time it is used for forming outbound trains in a classification process.

receiving yard subyard of a classification yard consisting of a number of parallel tracks that are connected to the line track at one end and to the lead track at the other; used to temporarily store arriving trains before brought to the lead track.

rolling stock all the vehicles moving on a railway, such as locomotives, passenger coaches, or freight cars; contrasts to the railway infrastructure.

shunting any movement of rolling stock other than timetabled train movements on lines; particularly includes sorting processes for railway rolling stock such as train classification.

shunting engine special railway locomotive built for shunting movements on railway yards, unlike passenger or freight locomotives used on lines.

sorting track classification track when it is used for rolling in cars in order to pull them out in a later step during a classification process.

train classification shunting process in which large amounts of railway freight cars are assembled into freight trains bound for some destination such as a station or railway yard.

unit train train in which all cars share the same origin and destination without any sorting requirements.