Multistage Methods for Freight Train Classification*

Riko Jacob[1], Peter Márton[2], Jens Maue[3], and Marc Nunkesser[3]

[1] Computer Science Department, TU München, Germany
`jacob@in.tum.de`

[2] Faculty of Management and Computer Science, University of Žilina, Slovakia
`peter.marton@fri.uniza.sk`

[3] Institute of Theoretical Computer Science, ETH Zürich, Switzerland
`{jens.maue|marc.nunkesser}@inf.ethz.ch`

Abstract. In this paper we study the train classification problem. Train classification basically is the process of rearranging the cars of a train in a specified order, which can be regarded as a special sorting problem. This sorting is done in a special railway installation called a classification yard, and a classification process is described by a classification schedule.

In this paper we develop a novel encoding of classification schedules, which allows characterizing train classification methods simply as classes of schedules. Applying this efficient encoding, we achieve a simpler, more precise analysis of well-known classification methods. Furthermore, we elaborate a valuable optimality condition inherent in our encoding, which we succesfully apply to obtain tight lower bounds for the length of schedules in general and to develop new classification methods. Finally, we present complexity results and algorithms to derive optimal schedules for several real-world settings. Together, our theoretical results provide a solid foundation for improving train classification in practice.

Keywords: rail cargo, shunting of rolling stock, sorting algorithms, train classification

1 Introduction

In real-world railways, a freight train consists of an engine and a number of railroad cars. Such a train does usually not commute between some fixed destinations, and its composition does not exist for a long period of time. Rather, freight trains are flexibly assembled from available cars on a daily basis according to customer demands, a process which is called *train classification*. Freight trains may be very long, their cars can have particular sorting requirements, or the trains may have to be assembled from cars originating from different stations, which causes the classification process requiring too many resources to

* This work was partially supported by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL). This work was partially supported by the Slovak grant foundation under grant No. 1-4057-07 (project "Agent oriented models of service systems").

be performed in a local station area. For this reason, there are centralized installations of railway tracks and switches, called *classification yards*, that exclusively serve the purpose of collecting, sorting, and reassembling freight cars in trains.

Train classification often is the bottleneck in freight transportation, and classification yards were designed decades ago to accommodate traffic requirements substantially different from those today. Expanding existing classification yards would require a lot of space, but, usually, there is simply no space available in a yard's periphery, particularly in densely populated areas such as Middle Europe. Furthermore, a redesign would make it necessary to reduce a yard's throughput or even completely stop it for some time period, which would impair the operation of a whole freight network since every yard represents a node in a densely connected network of yards. An obvious way to improve the performance of existing classification yards is to optimize the classification process itself. To this end, we revisit the most common classification methods and develop a completely novel encoding of train classification schedules. This encoding might be of general interest as it yields a consistent presentation of schedules and allows an efficient analysis and comparison of all existing—and any future—multistage classification methods. By means of our new encoding, we characterize optimal classification schedules and analyze the underlying algorithmic questions.

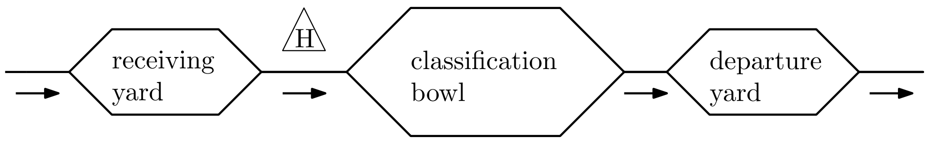

Fig. 1. A common classification yard with receiving yard, hump (H), classification bowl, and departure yard.

The typical layout of a classification yard, shown in Figure 1, consists of a receiving yard where incoming trains arrive, a classification bowl where they are sorted, and a departure yard where outgoing trains are formed. Many yards feature a *hump*, a rise in the ground, with a *hump track*. Cars are brought from the receiving yard to the hump track, on which they are slowly pushed over the hump, so that the cars accelerate by gravity and roll in on the tracks of the classification bowl. These yards are called *hump yards* in contrast to *flat yards* that require cars to be hauled by shunting engines. A typical classification bowl is shown in Figure 2(a), in which the classification tracks are connected to the departure yard at the end opposite the hump. Not all yards have receiving and departure tracks, some have single-ended classification bowls as in Figure 2(b), while others have a secondary hump at their opposite end as in Figure 2(c). However, almost all yards have the layout of Figure 2(b) as a substructure.

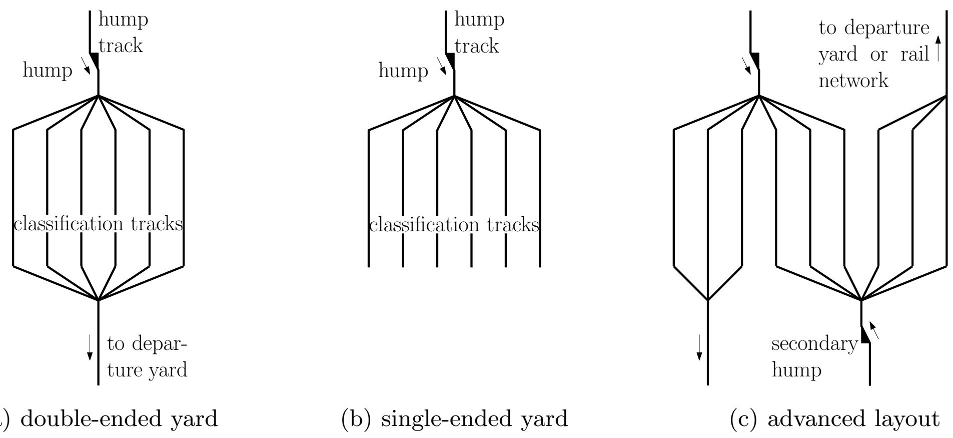

Fig. 2. Some common classification bowl layouts.

1.1 General Process of Train Classification

In the following description of the overall classification procedure, we consider a hump yard: inbound trains are collected in the receiving yard on a set of tracks called *receiving tracks*, from where they are moved to the hump track. There, the cars of the train are disconnected and the complete train is pushed over the hump by a shunting engine, sending the cars through a series of switches, separately guiding each car on a preassigned *classification track* of the classification bowl. This process is called a *roll-in operation.* Then, the actual sorting process is performed to produce outbound trains as outlined in the following two paragraphs. Finished trains are brought to the departure yard where they are finally picked up by freight locomotives to leave the classification yard.

Regarding the actual classification procedure, there are essentially two modes of operation for classification yards, which are typically performed in parallel or alternatingly: *single-stage* and *multistage sorting.* In single-stage sorting, each classification track usually corresponds to a common destination, such as a remote classification yard. Departing trains are built by collecting the cars from one or several tracks[1] and coupling them into trains that leave the bowl to the departure yard—if there is any. Single-stage sorting is normally performed for large volume traffic, e.g., traffic between classification yards, and the cars of the created trains are in arbitrary order.

For traffic directly going to its final destination, the order requirements for outgoing trains are more complex. Such trains usually serve many destinations, so single-stage sorting would require too many classification tracks. Furthermore, there may be only a few cars for each destination, so the classification tracks would not even be fully occupied, yielding a huge amount of idle track space. For both reasons, single-stage sorting is not applicable here; instead, multistage sorting is applied. This method requires more movement of cars

[1] In Europe it is common practice to collect each outbound train on a single track. In the U.S. the lengths of many freight trains exceed the length of the classification bowl. Therefore, outbound trains are collected from several tracks. The latter model is also assumed in [10].

but allows much more flexible and efficient use of the tracks than single-stage sorting. In multistage sorting, after the incoming trains have been pushed over the hump (*primary sorting*), a shunting engine repeatedly pulls back the cars from a given track (*pull-out operation*) over the hump on the hump track. These cars are then pushed over the hump again, so that again each car can be independently routed through the switches to any classification track. This process is iterated until all outgoing trains have been formed. This part of the sorting procedure is called *secondary sorting*. If a classification track is used only for receiving cars of an outgoing train, but the cars on it are never pulled back to the hump track, it is called *train formation track*.

1.2 Related Work

Multistage methods are presented from an engineering point of view in a number of publications from the 1950s and 1960s [2, 3, 13, 16, 21, 22, 25]. Krell [21, 22] compares the two basic multistage methods *sorting by train* and the *simultaneous method* togwther with two improvements of the latter, called *triangular sorting* and *geometric sorting*, respectively. (We will describe these four important methods in further detail in Section 4 by means of our representation of classification schedules introduced in Section 3.) Krell also includes an example showing how to deal with a restricted number of available classification tracks, which anticipates our generalized method for this kind of restriction as introduced in Section 5. Sorting by train and the simultaneous method had been described earlier in a different fashion by Flandorffer [16] and by Pentinga [25]. Boot [3] describes the operational constraints when the simultaneous method was introduced in France, Belgium, and The Netherlands. Other practical considerations regarding its real-world implementation, such as required yard layouts or arrival and departure times of freight trains, can be found in [22] and [13].

Some of these methods were again considered by Siddiqee [26] in 1972 and in a series of publications in the 1980s by Daganzo, Dowling, and Hall [6–9]. Siddiqee summarizes the main characteristics of the four methods mentioned above with regard to their applicability under different conditions [26]. The author recommends which method is suitable under which circumstances, taking into account the number and length of the tracks, the number and sorting requirements of outgoing trains, and the finishing times of the ougoing trains. In [9] the triangular method is analyzed with regard to the differences when applied in hump and flat yards. Triangular sorting and sorting by train are compared in a practical example in [6] including the concept of treating several outgoing trains as one, which the authors call *convoy formation* [6]. [7] and [8] introduce a variant of these two methods, called *dynamic blocking*, which reduces the number of required tracks for homogeneous [7] and heterogeneous traffic [8], respectively.

Baumann [2] explains the design aspects concerning multistage train formation for the example of the classification yard Zürich-Limmattal in Switzerland. The resulting layout features a secondary hump similar to Figure 2(c), which is, however, currently not used due to cost and organizational reasons [19].

For single-stage sorting, more theoretical research has been done than for multistage sorting in the last decade. An algorithmic single-stage sorting problem was studied by Dahlhaus et al. [10, 11]. For their train classification model, they give a notion of presortedness of the incoming train which is used to improve the classification process. Several degrees of freedom in the order requirement of the outbound train are regarded in [11], while finding an optimal schedule is shown to be NP-hard in [10] for the output requirement in which cars of the same type must appear consecutively but the order of types is free.

A systematic framework for classifying single- and also multistage classification methods is given by Hansmann and Zimmermann [18]. For the case of a limited number of classification tracks and an extended output requirement which handles several cars of the same type, they independently obtain the result we give in Section 5. Furthermore, the authors show for a specific multistage method that finding an optimal schedule is NP-hard for the output specification of [10] mentioned above.

The four methods mentioned in the first paragraph of this section are still the most widely applied multistage methods today, and they have been studied quite thoroughly in the engineering literature. We reconsider these methods from a theoretical point of view in Section 4, including a simpler description and more precise analysis.

1.3 Our Contribution

The main contribution of this paper is a novel encoding scheme for train classification schedules. This scheme offers an efficient and consistent representation of classification schedules, which we apply to analyze and prove certain properties of existing train classification methods. Secondly, we introduce the number of pull-out steps as an accurate yet very simple cost function for classification schedules, while we still take account of the traditional cost measure of bounding the number of roll-ins for every car as our secondary objective. Together with our encoding and a notion of presortedness for incoming trains, we obtain optimal schedules for sufficiently large yards, which implies in particular that it is impossible to outperform these schedules on real, restricted yards. We derive schedules for a restricted number of classification tracks. For tracks of limited length, we show the problem is NP-hard, give a 2-approximation and an exact algorithm for a special case. The main results are summarized in the following list:

- concise encoding of classification schedules (Section 3)
- provably tight lower bound for length of schedules (Theorem 2)
- application of novel encoding yielding a simpler, more precise analysis of well-known multistage methods (Section 4)
- polynomial-time algortihm to produce optimal schedules for yards with a limited number of tracks (Section 5)

– NP-hardness of general problem with tracks of bounded length (Section 6.1)
– polynomial-time 2-approximation for tracks of bounded length (Section 6.2)
– exact algorithm for tracks of bounded length if the incoming train is not assumed to be presorted (Section 6.3)

1.4 Outline

In the following section we introduce the above described problem and concepts formally, including our objective. In Section 3 we present our novel encoding for representing classification schedules, which allows us to concisely describe and analyze the above methods in Section 4. We proceed with analyses of new problem variants in Section 5 (restricted number of classification tracks) and Section 6 (tracks of restricted capacity). In Section 7 we briefly discuss secondary objectives and conclude with practical remarks and future work in Section 8.

2 Model and Notation

In this section we introduce the terminology and notation used in our model. We assume the common yard layout of a single- or double-ended classification bowl with a single hump as shown in Figure 2(b) and Figure 2(a), where the classification tracks are denoted by $\theta_1, \ldots, \theta_W$. We denote their number by W, the *width* of the yard. We further assume that all tracks have the same length and denote by $C_{\max}$ the *capacity* of the yard, i.e., the maximum number of cars that fit on a track.

Any car τ will be represented by some positive integer $\tau \in \mathbb{N}$ and a train T by an (ordered) k-tuple $T = (\tau_1, \ldots, \tau_k)$ of cars $\tau_i \in \mathbb{N}$, $i = 1, \ldots, k$. Note that the engine, depending on the situation, may be at either end of the train and is not included in this definition. The number of cars k will also be called the *length* of T. In our model, there is a set of ℓ *incoming trains* and m order specifications corresponding to m *outgoing trains*, and our goal is to sort the incoming cars according to the given specifications to obtain the outgoing trains. The sets of incoming and outgoing trains are together called a *classification task*, for which we make the following assumptions: for the ℓ incoming trains $T^k = (\tau^k_1, \ldots, \tau^k_{n'_k})$, $k = 1, \ldots, \ell$, where n'_k denotes the length of the kth incoming train and $n := \sum_{k=1}^{\ell} n'_k$ the total number of cars, we assume $\tau^k_j \in \{1, \ldots, n\}$ and that all cars are distinct. Without loss of generality (w.l.o.g.), we assume that concatenating the outgoing trains in their given order yields the sequence $(1, \ldots, n)$—otherwise, the incoming cars can be renamed; in other words, if n_k denotes the length of the kth outgoing train, $k = 1, \ldots, m$, the kth outgoing train is given by $(1 + \sum_{i=1}^{k-1} n_i, \ldots, \sum_{i=1}^{k} n_i)$. Hence, an ordered set of m outgoing trains can be defined uniquely by an m-tuple $n_1, \ldots, n_m$.

For any train $T = (\tau_1, \ldots, \tau_k)$, car τ_1 is called the *head* of T, and, for any pair of cars τ_i, τ_j of T with $i < j$, we say τ_i is *in front of* τ_j. For a train T located on the hump track, the head of T represents the car that is closest to the hump. For a train T located on some classification track, its head represents the car closest to the dead-end. Thus, the train in Figure 3(b) is represented by $(6, 1, 4, 2, 3, 5)$ and the train in Figure 3(j) by $(1, 2, 3, 4, 5, 6)$.

Any multistage sorting method consists of a sequence of alternating roll-in and pull-out steps. In order to specify a single pull-out step, it suffices to specify the classification track from which to pull out all cars. Note that always *all* cars on a track are pulled out; still, pulling only some cars can easily be achieved in this model by first pulling all cars and directly rolling in some of them to the same track. To fully specify a roll-in operation, however, a target track must be given for every car on the hump track. We call such a pair of operations a *sorting step* or simply *step*. An initial roll-in followed by a sequence of h sorting steps is called a *classification schedule* of *length* h. A classification schedule is called *valid* for a classification task if it transforms the given set of incoming trains into the set of outgoing trains. As mentioned before, our main objective will be the number of sorting steps, while the traditional objective, the number of cars rolled in, is studied in Section 7.

Definition 1 (Train Classification Problem). *Given a classification task of ℓ incoming trains $T^i = (\tau^i_1, \ldots, \tau^i_{n'_i})$, $i = 1, \ldots, \ell$, and the lengths $(n_1, \ldots, n_m)$ of the m outgoing trains, the train classification problem is the problem of finding a valid classification schedule of minimum length. We call such a classification schedule optimal.*

Note that, according to the definitions above, the term *length* may refer either to the number of sorting steps of a schedule or to the number of cars of a train. In the remainder of this paper, the respective meaning will always be clear from the context. Moreover, we will sometimes abbreviate statements referring to pull-out steps, such as abbreviating "the cars of a track are pulled out" to "a track is pulled (out)".

3 Encoding Classification Schedules

In this section we introduce an encoding of classification schedules by sets of bitstrings. Conversely, we show how to interpret such a set as a schedule, which yields a bijective relation between sets of bitstrings and classification schedules. Furthermore, a notion of presortedness is introduced for incoming trains, which allows one to deduce optimal schedules. We first consider the case of sorting a single incoming train into a single outgoing train in Section 3.1, which is extended to the general case in Section 3.2. Both sections assume classification yards without any restrictions on the number or capacities of the classfication tracks. (Restricted settings are considered in later sections.) This section concludes with practical notions on our concept of presortedness in Section 3.3.

3.1 Single train

We start by introducing a simplified view of the tracks. After a track has been emptied, cars may be sent to it in subsequent steps, so one physical track might be filled and emptied more than once during a classification procedure. We model this by introducing *logical tracks* that we define such that pull-out i is performed on logical track i. This means that the logical tracks are pulled out in the order $(1, 2, \ldots, h)$. For a classification schedule of length h, the mapping from the h logical tracks to the W physical tracks is given by a sequence $(\theta_{i_1}, \ldots, \theta_{i_h})$ of physical track names, called the *track sequence*.

The course of a single car τ can now be represented by an h-bit binary string $b = b_h \ldots b_1$, $b_i \in \{0, 1\}$, where $b_i = 1$ if and only if τ visits the i-th (logical) track. (In the following, these strings are interpreted as binary representations of non-negative integers with b_h being the *most* significant bit of b; in the usual depiction, this corresponds to using the tracks from right to left.) This representation uniquely defines the course of car τ: τ is pulled out in the i-th step if $b_i = 1$ simply because it has been sent there in some earlier step. Then, it is rolled in on the k-th track given by $k := \min_{j>i, b_j=1} j$, i.e., the lowest bit $b_k = 1$ left of i. If $b_j = 0$ for all $j > i$, then τ is sent to the train formation track of its target train. The track for the initial roll-in is given by the least significant bit b_i with $b_i = 1$. The complete schedule for a train of n cars can now be represented by a *binary encoding of length* h given by $B = (b^1, \ldots, b^n)$, a sequence of binary numbers (bitstrings), such that $b^j = b^j_h \ldots b^j_1$ encodes the course of the jth car, $j = 1, \ldots, n$. We will always use superscripts to refer to the bitstring of a car, e.g., b^j is the bitstring of the jth car of some train. Subscripts will refer to a single bit of some bitstring, e.g., the ith bit of some bitstring b is referred to by b_i.

An example is given in Figure 3, which shows a classification procedure and the binary representation of its schedule for a single incoming train of six cars. There are more classification tracks than schedule steps, so the above mentioned mapping from logical to physical tracks does not use any physical track more than once. Note that in our model the classification process is not yet finished in situations (e) or (g); a valid outgoing train is obtained only when the situation in (j) has been reached.

We will repeatedly use the following ordering of bitstrings in this section: let $b = b_{h-1} \ldots b_0$ and $b' = b'_{h'-1} \ldots b'_0$ be two bitstrings of lengths h and h', respectively, and let $x = \sum_{i=0}^{h-1} 2^i b_i$ and $x' = \sum_{i=0}^{h'-1} 2^i b'_i$ be the integers corresponding to these bitstrings. Then, if $x < x'$, we say that b is *smaller* than b', denoted by $b < b'$. In other words, we order bitstrings by the order of the integers they represent.

The following lemma shows how to read the binary representation of schedules: if two cars have different bitstrings, the car with the smaller bitstring will be located in the target train at a position closer to the head of the train. If two cars have the same bitstring, they will not swap their relative order.

Lemma 1. *Let $B = \{b^1, \ldots, b^n\}$ be the binary encoding of a classification schedule for an incoming train $T = (\tau_1, \ldots, \tau_n)$. Two cars τ_i and τ_j, $i < j$, swap their relative position if and only if $b^i > b^j$.*

Proof. If $b^j < b^i$, let k be the most significant index with $b^j_k = 0$ and $b^i_k = 1$. Car τ_i is sent to some track θ_{next} in sorting step k. As $b^j_\ell = b^i_\ell$ for all $\ell > k$, car τ_j was already sent

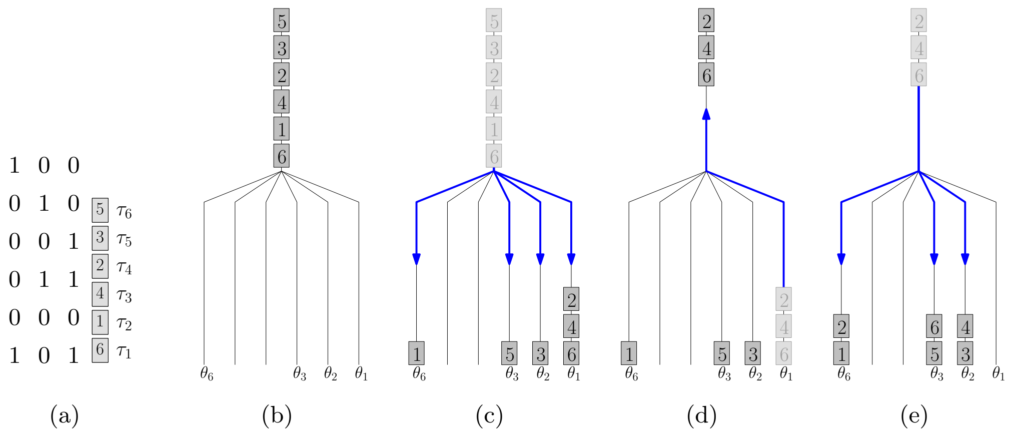
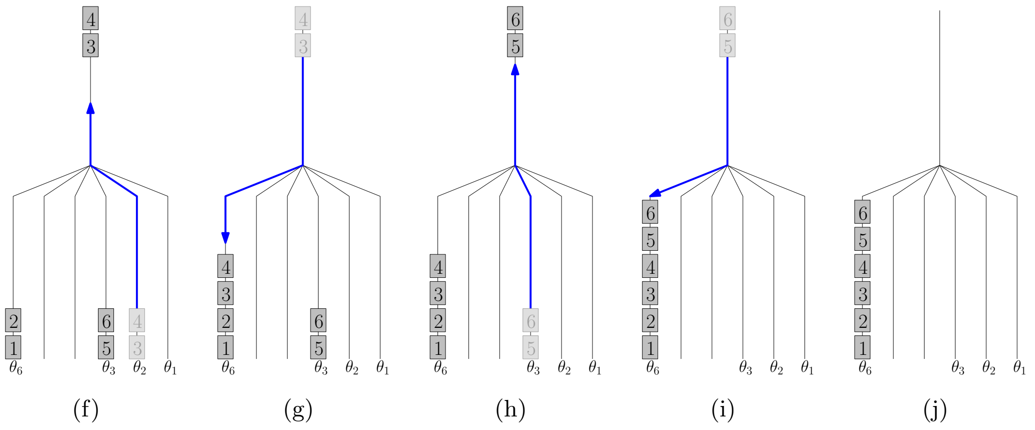

Fig. 3. A classification procedure for $h = 3$ and $n = 6$, using track θ_6 for the output train. The encoding is shown in (a), the input train in (b). (c)–(j) show the consecutive situations during the procedure, always pulling out the cars of the rightmost occupied track.

to θ_{next} in a previous step, so τ_j appears at a position in front of τ_i on θ_{next}. For the same reason, the two cars will not swap their relative order at any step later than k, so τ_j ends up on the output track at a position in front of τ_i in the outgoing train.

Conversely, if $b^j > b^i$, swapping the roles of b_i and b_j in the above argument yields that τ_i ends up in the outgoing train at a position in front of τ_i, so their order has not swappped. Finally, if $b^i = b^j$ the two cars will go exactly the same course and end up in the same order as in T. □

In the following theorem, we show that there is a bijection between valid classification schedules and binary encodings with a special property.

Theorem 1. *Let $T = (\tau_1, \ldots, \tau_n)$ be an incoming train, and let $B = (b^1, \ldots, b^n)$ be the binary encoding of a classification schedule with each b^i being a bitstring of length h, $i = 1, \ldots, n$. In a classification yard of sufficient width and capacity, B represents a valid* *schedule of length h for T if and only if the following properties are both satisfied:*

$$\text{For all } i,j \in \{1,\dots,n\} \text{ if } i<j \wedge \tau_i > \tau_j \text{ then } b^i > b^j \ . \tag{P1}$$

$$\text{If } b^i > b^j \text{ then } \tau_i \geq \tau_j \ . \tag{P2}$$

Proof. Property (P1) states that if two cars need to be swapped ($i < j$ and $\tau_i > \tau_j$) they are swapped, which follows from Lemma 1. If $b^i > b^j$ car τ_j ends up in front of car τ_i, irrespective of their prior order. This is correct if $\tau_i \geq \tau_j$. Therefore, by property (P1) cars that need to change their relative positions do so. Together, these two properties imply the validity of the schedule. □

As a consequence of the above theorem, an optimal schedule corresponds to a binary encoding B of minimum length h that satisfies properties (P1) and (P2). To construct an optimal schedule, we must take the shortest possible bitstrings, so we need to use as few different bitstrings as possible. Therefore, we want to know which cars of the incoming train can get the same bitstring, which leads to the following notion of presortedness.

Definition 2 (Chains). *Given an incoming train $T = (\tau_1,\dots,\tau_n)$, a pair of cars τ_j, τ_k is called a* break *if $j < k$ and $\tau_j = \tau_k + 1$; in other words, if two cars τ_j, τ_k appear consecutively in the outgoing train but are in the wrong order in the incoming train, they are called a break. The set of breaks of T uniquely decomposes each train into maximal subsequences of cars in the correct order; such a subsequence is called a* chain*, and the set of chains can be ordered ascendingly by their first elements.*

For example, train $T = (9,4,5,7,1,2,8,6,3)$ uniquely decomposes into the disjoint chains $c_1 = (1,2,3)$, $c_2 = (4,5,6)$, $c_3 = (7,8)$, and $c_4 = (9)$, which we call the *chain decomposition* of T.

By Definition 2 all cars of a chain are in the correct relative order with respect to the outgoing train, so, by Lemma 1, these cars can get the same bitstring. The following lemma shows how to assign bitstrings to chains.

Lemma 2. *Let $T = (\tau_1,\dots,\tau_n)$ be an incoming train and let $B = (b^1,\dots,b^n)$ be a valid schedule for T. If two cars $\tau_i < \tau_j$ belong to different chains, then $b^i < b^j$.*

Proof. If $i > j$, then τ_i and τ_j must swap their relative order since $\tau_i < \tau_j$, so $b^i < b^j$ must be satisfied by Lemma 1. If $i < j$, then there is a car τ_x with $\tau_i < \tau_x < \tau_j$ and either $x < i$ or $x > j$. (Otherwise, τ_i and τ_j would be in the same chain.) If $x < i$, then $b^i < b^x$ and $b^j \geq b^x$ by Lemma 1, so $b^i < b^j$. Similarly, if $y > j$, then $b^j > b^y$ and $b^i \leq b^y$ by Lemma 1, so $b^i < b^j$. □

The above assignment of bitstrings to chains finally yields the main result of this section, which is summarized in the following theorem.

Theorem 2 (optimal schedules). *Let $T = (\tau_1 \dots \tau_n)$ be a train of length n and let c be its number of chains. In a classification yard of sufficient width and capacity, the optimal classification schedule B for T has length $h = \lceil \log_2 c \rceil$.*

Proof. As shown above, in a valid schedule cars of the same chain can be assigned the same bitstring. Hence, c bitstrings are sufficient for a valid assignment of bitstrings to cars, so $h \leq \lceil \log_2 c \rceil$ as there are 2^h different bitstrings of length h. Conversely, by Lemma 2, cars of different chains must be assigned different bitstrings. Thus, at least c different bitstrings are required for a valid schedule, so the length of any valid schedule satisfies $h \geq \lceil \log_2 c \rceil$. □

This result can be extended to more complex objective functions as explained in Section 7.

3.2 Multiple Trains

Any real-world classification task involves multiple incoming and multiple outgoing trains. Once the order of the incoming trains has been determined, such a classification task can be solved by separately solving, for each outgoing train, one classification task of a single incoming and single outgoing train. The individual solutions can then just be appended, which is shown in the following section. In the subsequent section, we deal with choosing an optimal order of multiple incoming trains.

Multiple Outgoing Trains In the following theorem, the sequence of incoming trains in a fixed order is regarded as a single incoming train.

Theorem 3. *For a classification yard of unrestricted width and capacity, let $T = (\tau_1, \ldots, \tau_n)$ be an incoming train, and let m outgoing trains be given by their lengths $(n_1, \ldots, n_m)$. Furthermore, let T_i be the subsequence of T corresponding to the ith outgoing train, $i = 1, \ldots, m$, and let B_i be the optimal schedule for the task of sorting T_i into the ith outgoing train, $i = 1, \ldots, m$. The length $|B|$ of an optimal schedule B for the sorting task given by T and $(n_1, \ldots, n_m)$ is $|B| = \max_{1 \leq i \leq m} |B_i|$.*

Proof. Every outgoing train has a separate train formation track, so if two cars belong to different chains with respect to different outgoing trains, their relative order is irrelevant during the whole classification process. Hence, they can be assigned the same bitstring. In the optimal schedule B, every car can thus be assigned to the bitstring it is assigned to in its respective schedule B_i, artificially increasing the length of the bitstring to $\max_{1 \leq i \leq m} |B_i|$ by adding leading zeros. □

The theorem statement does not change for classification yards with a restricted number of tracks as discussed in Section 5. For tracks of restricted capacity as discussed in Section 6, however, the theorem does not hold.

Multiple Incoming Trains We have assumed a fixed order of the incoming trains so far. This assumption is reasonable in most practical cases where the incoming trains arrive distributed over time. Assuming a single outgoing train, changing the order of incoming trains, however, may change the number of breaks in the resulting sequence of incoming cars. Hence, if the order can be chosen freely, the problem of choosing an optimal order of incoming trains arises.

Definition 3. *Given ℓ incoming trains $T_i = (\tau^i_1, \dots, \tau^i_{n'_i})$, $i = 1, \dots, \ell$,* OPT-PERM *is defined as the problem of finding a permutation π that minimizes the number of breaks in the concatenation $T_{\pi(1)} \cdots T_{\pi(\ell)}$ of the incoming trains.*

Below, Lemma 3 shows the problem OPT-PERM is equivalent to a special minimum feedback arc set problem. The general minimum feedback arc set problem (MinFAS) is defined in the following way: given a directed graph $G = (V, E)$, find a linear arrangement $\pi(V)$ of V that minimizes the number of edges $(u, v) \in E$ with $\pi(u) > \pi(v)$. Put less formally, find an arrangement of nodes that minimizes the number of backward edges. A detailed survey of feedback arc set problems can be found in [15].

For multiple incoming trains, breaks may occur between two cars of the same incoming train (which we will call internal breaks) or between two cars of different incoming trains (external breaks). This is summarized in the following definition which is used in the proof of Lemma 3 below.

Definition 4. *Let $T = (T^1 \cdots T^\ell)$ be the concatenation of ℓ incoming trains $T^i = (\tau^i_1, \dots, \tau^i_{n'_i})$, $i = 1, \dots, \ell$, and let $p = (\tau^i_x, \tau^j_y)$, $i, j \in \{1, \dots, \ell\}$, be a break of T, i.e., $\tau^i_x = \tau^j_y + 1$ and either $i < j$ or $i = j$ and $x < y$. If $i < j$, then p is called an* external break *between T^i and T^j. If $i = j$, then p is called an* internal break *of T^i.*

The following lemma finally shows the equivalence between OPT-PERM and MinFAS in a special class of graphs.

Lemma 3. *There is a one-to-one correspondence between OPT-PERM and MinFAS for directed multigraphs the edges of which form an Eulerian path.*

Proof. We first show how to transform OPT-PERM from Definition 3 into a minimum feedback arc set instance $G = (V, E)$. Each incoming train T^i is mapped to a node $n(T^i)$. For each pair of cars $\tau_k \in T^\alpha, \tau_j = \tau_k + 1 \in T^\beta$, we add a directed edge $(n(T^\alpha), n(T^\beta))$ to E. Traversing the sequence of cars in this way, we add exactly $n - 1$ edges, potentially self-loops, to G, and these edges form an Eulerian path by the construction. We remove the self-loops and still have an Eulerian path. If π is any permutation of incoming trains and T denotes the resulting incoming sequence of cars, the number of external breaks of T equals the number of arcs pointing backwards in the linear arrangement $\pi(V)$. Since the number of internal breaks of T does not change by altering the order of incoming trains, a permutation of trains minimizing the total number of breaks corresponds to an arrangement of nodes that minimizes the number of backward arcs.

For the other direction, the following construction obviously transforms any multigraph with an Eulerian path into an instance of OPT-PERM with the same relation of the objective functions: for each node $n \in V$, we introduce an incoming train $T(n)$. Then, we walk along the Eulerian path $P = (n_{i_1}, \dots, n_{i_m})$ and add for each $n_{i_j} \in P$ a car j to train $T(n_{i_j})$. □

To the best of our knowledge, the complexity status of MinFAS in such graphs is open. By a lemma of Newman, Chen, and Lovász [24, Theorem 4], however, a polynomial algorithm for OPT-PERM would lead to a $\frac{16}{9}$-approximation algorithm for MinFAS in general graphs, substantially improving over the currently best known $O(\log n \log\log n)$-approximation [14].

3.3 Groups and Blocks

Another important practical complication concerns groups and blocks that occur in real-world classification. These concepts are closely connected to presortedness of incoming trains, to sorting specifications of outgoing trains, and thus to our notion of chains. We will see that these aspects do not significantly change the results above.

A *group* is a set of cars with the same destination, typically the same customer. These cars can come from different incoming trains but always go to the same outgoing train, where they occur consecutively. The order of cars in a group is irrelevant. Still, the order of the groups in the outbound trains is fixed. A *block* is a set of cars (or groups) that take a common itinerary over potentially many classification yards. A block is not broken up at intermediate classification yards. The associated blocking and makeup problems are out of the scope of this paper; see [1, 5] for references. Blocking is particularly advantageous in large countries like the U.S. and usually not applied in European freight systems [4].

Both concepts are related in the following way: before a block starts its itinerary over several yards, it must be built at some origin classification yard. If the block is to be built by multistage sorting, it can be regarded as a group of an outgoing train and can be formed in the same way. If a block arrives at an intermediate destination, it is not broken up and can be regarded as a weighted car for the classification task. Finally, a block arriving at the final destination yard of its itinerary is broken up like an incoming train and has no influence on the classification schedule. Thus, for our purposes, both groups and blocks lead to exactly the same complication: some cars must be sorted in a sequence the order of which is irrelevant. Therefore, w.l.o.g., we will only refer to groups below.

To adapt our model we simply drop the constraint that all cars $(\tau_1, \ldots, \tau_n)$ of a train must be distinct. It is easy to check that with this definition Lemma 1 and Theorem 1 still hold. Theorem 1 applied to trains with groups, however, leads to a different notion of chains than the one in Definition 2.

As an analogue of Definition 2 and Lemma 2 we redefine chain decompositions with our goal of extracting optimal binary encodings from them:

Definition 5. *Given a train $T = (\tau_1, \ldots, \tau_n)$ with not necessarily distinct τ_i, a* chain decomposition *$\Xi = (c_1, \ldots, c_k)$ is a partition of T into subsequences, such that the following resulting classification schedule is valid: Only the binary representations of $i-1$, $1 \leq i \leq k$, are used. All cars of chain c_i are assigned the binary code $i-1$, and this binary encoding is called $B(\Xi)$. We call Ξ* optimal *if $B(\Xi)$ is optimal.*

Observe that there is no longer a single optimal chain decomposition. For example, for the train $(3, 1, 4, 1, 2, 2, 3, 5, 4)$ both $(1, 1, 2, 2), (3, 3, 4), (4, 5)$ and $(1, 1, 2, 2, 3), (3, 4, 4), (5)$ are optimal. The second chain decomposition in the example has been obtained in the following greedy way: The first chain c_1 contains all possible cars with $\tau_. = 1$. All cars with $\tau_. = 2$ can get the same bitstring by Lemma 1. All occur after the last $\tau_. = 1$. They are included in c_1. Not all cars with $\tau_. = 3$ can get the same bitstring, because the first 3 needs to be swapped with the 1s and 2s. Still, the second 3 can get the bitstring of the first chain because it does not need to be swapped with the 1s or the 2s. It follows that $(1, 1, 2, 2, 3)$ is the longest possible sequence of cars that can get the same bitstring and contains the smallest car, here $\tau_. = 1$. The remaining chains can be obtained by iterating this procedure on the remaining cars. Let us call the resulting chain decomposition the *greedy chain decomposition*. We now show the analogue of Theorem 2 for groups.

Theorem 4. *(greedy chain decomposition is optimal) Let $T = (\tau_1, \ldots, \tau_n)$ be a train with not necessarily distinct τ_i. The greedy chain decomposition $\Xi_g = (c_1, \ldots, c_k)$ of T leads to an optimal classification schedule of length $\lceil \log_2 k \rceil$ encoded as $B(\Xi)$ for a classification yard of unrestricted width and capacity.*

Proof. Assume for a contradiction that there is another chain decomposition with fewer chains. We want to compare chains car by car and therefore distinguish the cars by indices. The above example becomes $(3_1, 1_1, 4_1, 1_2, 2_2, 2_3, 3_2, 5, 4_2)$. Select the optimal chain decomposition $\Xi_{\text{opt}} = (d_1, \ldots, d_{k'})$, with $k' < k$ that has the longest identical start sequence of chains $(c_1 = d_1, c_2 = d_2, \ldots)$ among all optimal chain decompositions. Again, for the above example we have $(1_1, 2_2) \neq (1_2, 2_2)$. Now consider the first pair of chains (c_x, d_x), where Ξ_{g} and Ξ_{opt} differ. From the greedy construction and the fact that the two decompositions agree up to c_x and d_x it follows that c_x must be longer than d_x. In fact, $d_x \subset c_x$ because all cars in d_x can be included in c_x independent of the cars in $c_x \setminus d_x$. We modify Ξ_{opt} to Ξ'_{opt} by replacing d_x by c_x and removing the cars in $c_x \setminus d_x$ from the rest of the chains in Ξ_{opt}. Note that we cannot invalidate a chain by removing cars from it. Now Ξ'_{opt} is a chain decomposition and agrees with Ξ_{g} on one more chain in contradiction to the assumption. □

4 Multistage Classification Methods

In this section we describe the multistage classification methods mentioned in Section 1.2 by means of the encoding introduced in Section 3. First, we explain how classification methods can be regarded as classes of schedules. Then, sorting by train and the basic version of simultaneous sorting are analyzed in Section 4.2, followed by two variants of the latter called triangular and geometric sorting in Section 4.3.

4.1 Classification Methods and Classes of Schedules

All schedules that follow a common classification method share common properties. For the example of triangular sorting—an explanation of this method 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 schedule can be represented by an encoding according to Section 3. Since this encoding contains the properties of the schedule, it contains all properties of the classification method that the schedule implements. Therefore, every classification method can simply be regarded as a *class of classification schedules* with certain properties, so every classification method corresponds to a *class of encodings*.

In the remainder of Section 4, we assume that we are given m outgoing trains by their lengths $n_1, \ldots, n_m$ as explained in Section 2. Let $g_1, \ldots, g_m$ be the respective numbers of groups in these trains. Define $g_{\max} = \max\limits_{1 \le i \le m} g_i$ and $g_{\min} = \min\limits_{1 \le i \le m} g_i$.

4.2 Basic Multistage Methods

The multistage methods applied today can be categorized into two general classes: *sorting by train* [6, 21, 26] and *simultaneous sorting* [16, 21, 25, 26], which have been extensively described in the literature. Using our encoding, we achieve a more efficient description and analysis of both methods in this section.

Sorting by Train Sorting by train comprises the following two stages. First, inbound cars are separated according to their outgoing trains by sending all cars of a common outgoing train to the same track. Second, the resulting unordered trains are processed successively: 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. Finally, the groups of cars are moved from the tracks in the required order and coupled to form an outgoing train. In double-ended yards this can be done from the opposite end of the yard. The process continues with the next train. For the rest of this paper, the train formation tracks will not appear in the encoding since they are implicitly given.

The corresponding class of encodings is given as follows: for the encoding $b_h \ldots b_1$ of any car in the ℓth group of the kth outgoing train, $b_i = 1$ only for $i = k + \sum_{j=1}^{k-1} g_j$ (corresponding to the initial roll-in) and $i = k + \sum_{j=1}^{k-1} g_j + \ell$ (corresponding to the second stage).

The length h of this schedule is given by $h = m + \sum_{i=1}^{m} g_i$. This method occupies exactly m classification tracks after the first stage, so the total number of tracks is at least $m + g_{\min} - 1$, and at most $m + g_{\max} - 1$, while the latter number is tight if the first train processed in the second stage has $g_{\max}$ groups.

Sorting by train is also called *initial grouping according to outbound trains* [26]. The corresponding names used in the German literature are *Ordnungsgruppenverfahren* [21] and *Staffelverfahren*.

Simultaneous Sorting Unlike sorting by train, the first stage of the two-stage method simultaneous sorting sorts according to the cars' groups in the outgoing train. In the encoding of a schedule of this class, this step forces $b_i = 1$ for every car of the ith group of any train. In the second stage, the cars are sorted according to their target trains: the tracks are successively pulled out 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 outgoing train to the same train formation track. As mentioned before, this last roll-in is not explicitly given in our encoding.

Simultaneous sorting minimizes the number of cars rolled in, which must be paid for by a number $g_{\max}$ of sorting steps as there is one pull-out for every group, a number which is maximal among all variants of simultaneous sorting for an unrestricted classification yard. 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 for train formation, and at least $g_{\min} + m - 1$ tracks are required: Even after renaming the groups such that all outgoing trains have a group $g_{\max}$ and use the small numbers as little as possible, pulling the first of the last $g_{\min}$ tracks to be pulled in the second stage forces starting the formation of all m outgoing trains (if not yet started), so exactly $g_{\min} + m - 1$ tracks are occupied at this point.

In contrast to sorting by train, the formation of all outgoing trains is performed simultaneously, which explains the method's name. Simultaneous sorting is also called the *simultaneous method*, *simultaneous marshalling* [25], *sorting by block* [6], or *initial grouping according to subscript* [26]. The corresponding German terms are *Simultanverfahren* and furthermore *Elementarsystem* to explicitly refer to the basic version described in this section [21].

Note that in the above description cars are never guided to a train formation track during primary sorting. This may be desired for operational reasons and is even necessary if the train formation tracks cannot be accessed from the primary hump such as for a layout as shown on the right of Figure 2(c). If the train formation tracks can be accessed from the primary hump, however, the schedule becomes one step shorter.

4.3 Variants of Simultaneous Sorting

In the 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* has been considered in [6, 9, 16, 21, 25, 26]. This method is a superclass of simultaneous sorting that is given by allowing at most three roll-in operations for each car (including the final roll-in of a car to its outgoing train). For the schedule encoding, this yields a restriction of $b_i = 1$ for at most two indices $i = 1, \ldots, h$ in the bitstring of every car.

For this method Krell [21] gives an upper bound of $\frac{1}{2}h(h+1)$ on the maximum number of groups $g_{\max}$ of an outgoing train that can be sorted in h steps. This result can be reformulated in terms of chains yielding a better bound in general. If $c_1, \ldots, c_m$ denote the respective numbers of chains of the trains and $c_{\max} = \max_{1 \leq i \leq m} c_i$, for a sufficiently large classification yard, classifying by triangular sorting can be done within h sorting steps if $c_{\max} \leq \frac{1}{2}h(h+1)$. This follows immediately by our encoding: the number of distinct bitstrings $b_h \ldots b_1$ of length h and $b_i = 1$ for at most two different indices $i \in \{1, \ldots, h\}$ is given by $\binom{h}{1} + \binom{h}{2} = \binom{h+1}{2}$, and the required number of distinct bitstrings is not greater than the maximum number of chains by Lemma 2. Conversely, if some outgoing train has a number of chains $c_{\max}$, we must choose a set of $c_{\max}$ bitstrings with $b_i = 1$ for at most two different $i \in \{1, \ldots, h\}$, which yields a feasible schedule of length $h = \lceil \sqrt{2c_{\max}} - \frac{1}{2} \rceil$ that can be applied in a sufficiently large classification yard. The method can be generalized to any restriction 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 [21], triangular sorting is called *Vorwärtssortierung bei höchstens zweimaligem Ablauf.*

Geometric Sorting The method of *geometric sorting* [16, 21, 25, 26] is derived from simultaneous sorting by dropping the restriction on the number of roll-ins completely, which 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 [21]. Note that this number differs from 2^h by one as cars are not allowed to be sent to the train formation track in the initial roll-in (for some practical reason), which corresponds to disallowing the all-zero bitstring in our encoding. In combination with our notion of chains this exactly yields the class of schedules of Theorem 2 with a bound of $c_{\max} \leq 2^h - 1$, where $c_{\max}$ denotes the maximum number of chains in any outgoing train.

Considering the special case of a single outgoing train of $2^k - 1$ groups for some integer $k > 0$, the initial roll-in sends $2k - i$ 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 [21].

As mentioned before, geometric sorting minimizes the number of sorting steps, assuming the number and capacities of tracks are sufficiently large. If this cannot be assumed, the simultaneous sorting variants of the following sections should be considered.

5 Restricted Number of Classification Tracks

In this section we consider schedules for classification yards with a restricted number W of classification tracks. In Section 3 we derived schedules of length $h = \lceil \log_2 c \rceil$ for a single incoming train with c chains. These schedules generally require one classification track for every sorting step, i.e., $W \geq \lceil \log_2 c \rceil$ must hold. (Note that the train formation track is not included in the number W.) However, such a schedule is not valid in general

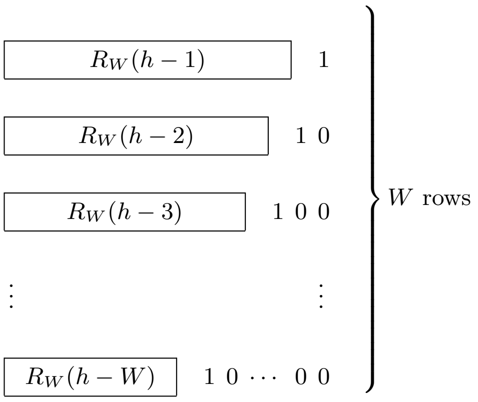

Fig. 4. Recursive procedure for deriving all valid bitstrings of length h for the round robin track sequence in a yard of W tracks.

if $W < \lceil \log_2 c \rceil$, which can easily be seen from the example in Figure 3: the illustrated schedule has a length of $h = 3$ and uses $W = 3$ classification tracks. If there are only two tracks, car $\tau_6 = 5$ cannot be sent to track Θ_3 at the initial roll-in since that track does not exist. Hence, this schedule is not valid for $W = 2$.

Still, there always exists a valid schedule for every $W \geq 2$, which we will show below. We consider a classification yard with a restricted number of W classification tracks $\Theta_1, \ldots, \Theta_W$ of sufficient capacity. We further assume a single incoming and a single outgoing train since this implicitly solves the general case of multiple incoming and outgoing trains as already described in Observation 3. In addition to the set of bitstrings B, the order in which the classification tracks are pulled out must be specified in a schedule of this setting as $h > W$. This is done by introducing a sequence of *logical tracks* $\Theta_{i_1}, \ldots, \Theta_{i_h}$ which maps sorting steps to physical tracks, i.e., in the kth step we pull out track Θ_{i_k}. We always use a "round robin" track sequence, i.e., in the kth step we pull out track $\Theta_{(k \bmod W)}$, where Θ_0 is treated as Θ_W. We later show that this strategy is optimal.

Basically, some of the 2^h bitstrings of length h are not valid due to the restricted number of tracks, so they cannot be used in the assignment from cars to bitstrings. The following lemma shows how many valid bitstrings of length h there are for the round robin logical track sequence. This number implies a lower bound for the number of chains that can be classified within h steps. Theorem 5 finally shows this number is optimal.

Lemma 4. *For a classification yard with W classification tracks of sufficient capacity, and additional dedicated train formation tracks for the outgoing trains, the number $R_W(h)$ of valid bitstrings of length h for the round robin track sequence is given by the following recurrence equation:*

$$R_W(h) := \begin{cases} 2^h & \text{for} \quad h \leq W \\ 1 + \sum\limits_{i=h-W}^{h-1} R_W(i) & \text{for} \quad h > W \end{cases}$$

Proof. Let $b = b_{h-1} \dots b_0$ be a bitstring of length h. Whenever a car is rolled in, it must be sent to one of the W physical tracks. In the kth pull-out step of the round robin strategy, these tracks always correspond to the W next pull-out steps, i.e., to the W next logical tracks $\Theta_{i_{k+1}}, \dots, \Theta_{i+k+W}$. (The inital roll-in can be regarded as following the 0th pull-out step.) Therefore, if $b_{k-1} = 1$, $b_{k-1+i} = 1$ must hold for at least one bit $i = 1, \dots, W$; in other words, for b to be valid, there must not be a sequence of W consecutive bits all equal to zero. However, since a car can always be sent to the train formation track, a *leading* sequence of W or more zeros is still allowed. If $h \leq W$, a non-leading sequence of W zeros cannot occur, so $R_W(h) = 2^h$ in this case.

For longer bitstrings consider Figure 4: at the initial roll-in there are W choices for any car, so $b_i = 1$ must hold for some $i = 0, \dots, W-1$. Now, let i be the smallest such index in b, corresponding to one of the W rows in Figure 4. After the pull-out corresponding to $b_i = 1$, there are again W choices of logical tracks to which to send a car assigned to b. Additionally, there is the possibility to send the car directly to the dedicated track for the outgoing train. The number of bitstrings starting with $b_i \dots b_0 = 10 \dots 0$ is $R_W(h-i-1)$ for this value of i. Summing over all W initial such sequences, corresponding to the rows in Figure 4, we obtain $R_W(h) = 1 + \sum_{i=h-W}^{h-1} R_W(i)$. □

The variation of the problem without dedicated tracks for the outgoing track has a similar solution. The only difference is that leading zeros are forbidden as well, which changes the recursion slightly:

Lemma 5. *For a classification yard with W classification tracks of sufficient capacity, where the outgoing train must be formed on the classification tracks, the number $R_W(h)$ of valid bitstrings of length h for the round robin track sequence is given by the following recurrence equation:*

$$R_W(h) := \begin{cases} 2^h & \text{for} \quad h < W \\ \sum\limits_{i=h-W}^{h-1} R_W(i) & \text{for} \quad h \geq W \end{cases}$$

Note that in this case the number $R_W(h)$ is a generalization of the Fibonacci numbers. In particular, for $W = 2$, we get $R_2(h) = F_{h+2}$ where F_h is the hth Fibonacci number. The proof of Lemma 4 yields a recursive procedure to derive optimal schedules, which is applied in the following theorem.

Theorem 5. *Given a classification yard with W classification tracks of sufficient capacity, an incoming train with c chains can be classified within h steps if and only if $c \leq R_W(h)$. The corresponding classification schedule can be constructed in linear time (in the size of the schedule).*

Proof. The forward implication of the first statement immediately follows from Lemma 4. For the reverse implication, we show round robin is optimal by induction. For the base case of $h \leq W$, the statement obviously holds. Now, assume the maximum number of bitstrings of length h (for the best possible track sequence) is $R_W(h')$ for all $h' < h$. Take an optimal track sequence and set of bitstrings of length h. These bitstrings can be divided into at most W classes by their second occurrence of a 1 (the positions depend on the track sequence). If the classes are ordered according to this position, the first class contains bitstrings of length at most $h-1$, the second of length at most $h-2$, and so on. Hence, the number of bitstrings in the classes is bounded by $R_W(h-1)$, $R_W(h-2)$, and so on. (This holds even if the different classes were allowed to have different track sequences.) Therefore, at most $R_W(h)$ bitstrings are possible. Hence, the round robin logical track sequence is optimal and no more than $R_W(h)$ chains can classified within h steps.

The proof of Lemma 4 yields a recursive procedure to construct a maximal set of valid bitstrings of length h in time linear in the size of the set. The procedure can be applied just partially in order to obtain an ordered set of c bitstrings of length h that present the schedule in time linear in the size of the schedule. $\square$

As mentioned before, Hansmann and Zimmermann independently obtain the same result in [18]. Moreover, a detailed example for this setting is already given in [21], though the description is quite cumbersome as there is no efficient representation of classification schedules. From the example, the maximum number of cars that can be sorted for a number of given tracks is correctly deduced, but no proof is given.

6 Restricted Track Capacities

In Section 3 we derived optimal schedules for the simplified case of classification tracks of unrestricted length. Real-world classification yards clearly have tracks whose length is bounded. With this additional constraint, the problem of finding an optimal classification schedule is NP-hard, which is shown in Section 6.1. For the same setting, we derive a factor-two approximation algorithm in Section 6.2. In the special case that no assumptions about the order of the cars in the incoming train is justified, an exact polynomial-time algorithm is given in Section 6.3.

6.1 General Case

As mentioned above, assuming bounded track capacities for the classification tracks yields an NP-hard problem. The bound on the track capacities is formalized as follows: All tracks have a bounded capacity of C, i.e., they can accommodate at most C cars. This capacity constraint is summarized in the following equation, where n is the number of cars, h the length of the schedule, and b_i^j denotes the i-th bit of the bitstring of the j-th car:

$$\sum_{j=1}^{n} b_i^j \leq C, \quad i = 0, \ldots, h-1. \tag{1}$$

The constraint does not apply to the train formation tracks, from which we furthermore do not allow any pull-outs.

Multiple Outgoing Trains The following theorem shows that the problem variant with multiple outgoing trains and an unrestricted number of classification tracks with restricted capacities is NP-hard. As a consequence, the train classification problem is NP-hard in its most general variant.

Theorem 6. *For a classification yard with classification tracks of restricted capacities, finding the optimal classification schedule for a single incoming and multiple outgoing trains is an NP-hard problem.*

Proof. By reduction from the NP complete problem "Not ALL Equal 3-SAT" (NAE3SAT) [17, LO3]. The input of NAE3SAT consists of n variables and m clauses, each of length 3. The problem is to decide whether there is a truth assignment such that each clause has at least one true and at least one false literal. Given an instance of NAE3SAT, we first construct an instance of $2n$ incoming trains that are to be sorted into $2n$ outgoing trains without any interaction between the trains, i.e., the ith incoming train has cars only for the ith outgoing train. Note that even though there are multiple incoming trains their order is irrelevant, because there is a one-to-one correspondence of incoming to outgoing trains (this is in contrast to the general situation discussed in Lemma 3). For this reason the incoming trains can be concatenated in any order into a single incoming train. For ease of exposition, we start the proof by making two assumptions, and show later that these can be easily enforced. First, each car can be part of at most one additional roll-in. Second, we can have individual capacity bounds for all logical tracks.

The main idea of the proof is to allow a given number $M = 4n + 2m$ of steps and thus logical tracks and to let all incoming trains have exactly $M - 1$ chains. Observe that if two chains of the same train end up on the same track they must be in the wrong order, which necessitates an additional roll-in incontradiction to the first assumption. It follows that at most one of the chains of each train can be split or a single logical track can be left unused. The transformation enforces the latter possibility for all trains.

From this construction it follows that the "local decisions" we can encode are for each train, which track should be left unused. With the help of additional start and end gadgets, the transformation forces the trains to have this unused track on the first or last logical track, which encodes true and false. These true and false settings for the variables are used in a middle part of the logical tracks that represents the clauses. In this part, the capacity restriction on the tracks leads to the not all equal constraint on the clauses.

We proceed to show how to use this idea in the transformation and give an example in Figure 5. First, for the incoming trains it is enough[4] to specify the length of each of their chains, instead of giving the full sequence that leads to these chains. For example we will define a train as $[1, 4, 2]$ by its sequence of chains (*chain sequence*) and ignore whether this comes from an incoming train $(2, 6, 1, 3, 4, 7, 5)$ or $(6, 2, 3, 1, 4, 5, 7)$. Chains and logical tracks are tightly connected because cars are pulled only once. For example, if there is

[4] All chain lengths will be polynomial in n and m. Therefore, this is only a matter of convenience and does not change the result for a car by car input encoding.

precisely one chain more than there are tracks, the first chain goes directly to the output track, the second on logical track one, and so forth. By the convention of numbering logical tracks from right to left (because of the relation to binary numbers), it is convenient to reverse the order in the chain sequence and to omit the first chain that is rolled-in directly. Hence, we write a *compact chain sequence* as $\langle 2, 5, 3\rangle$ to mean the chain sequence $[1, 3, 5, 2]$ and an incoming train like $(10, 11, 5, 6, 7, 8, 9, 2, 3, 4, 1)$.

$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$

	x_1	$\bar{x}_1$	x_2	$\bar{x}_2$	x_3	$\bar{x}_3$	C_1^+	C_1^-	C_2^+	C_2^-	x_1	$\bar{x}_1$	x_2	$\bar{x}_2$	x_3	$\bar{x}_3$
x_1	k	1	1	1	1	1	k	1	1	1	k	1	1	1	1	
$\bar{x}_1$		k	1	1	1	1	1	1	1	k	1	k	1	1	1	1
x_2		1	1	k	1	1	1	1	1	k	1	1	1	k	1	1
$\bar{x}_2$	1	1	k	1	1	1	k	1	1	1	1	1	k	1	1	
x_3		1	1	1	1	k	1	k	1	1	1	1	1	1	1	k
$\bar{x}_3$	1	1	1	1	k	1	1	1	k	1	1	1	1	1	k	

Fig. 5. Sketch of the transformation for an example with two clauses on three variables.

Each compact chain sequence has $4n + 2m - 1$ chains, which correspond to a *start variable part* of length $2n$, followed by a *clause part* of length $2m$ and an *end variable part* of length $2n - 1$. There are $2n$ chain sequences, one for each literal. All chains have either length k (*ON*) or length 1 (*OFF*). The purpose of the start and end part of the chain sequences is to force the gap into these parts. This is achieved by defining the start and end parts of both x_i and $\bar{x}_i$ as follows:

$$\langle \overbrace{\underbrace{1, 1}_{\text{one pair/variable}}, \ldots, \underbrace{k, 1}_{\text{pair } i}, 1, 1, \ldots,}^{\text{start part}} \overbrace{\ldots}^{\text{clause part}} \overbrace{, 1, 1, \ldots, \underbrace{k, 1}_{\text{pair } i}, 1, 1, \ldots, 1, 1}^{\text{end part}} \rangle$$

Both sequences have length $M - 1$ together with the clause part that remains to be specified.

The first $2n$ logical tracks and the last $2n$ logical tracks have all capacity $2n + k - 1$, except for the first and the last track which have both capacity $n + k - 1$. The total capacity of the first $2n$ positions of all chain sequences exceeds the total available capacity for the start part by n, and the same holds for the end part. This situation forces at least n gaps in the start part and at least n gaps in the end part, thus exactly n gaps in both parts. Having these identical subsequences for a variable and its negation enforces, together with the capacity bound, that for each variable either there is a gap in the start part of the chain sequence for x_i and in the end part of the one for $\bar{x}_i$ or vice versa. Thus, we can think of the chain sequences for variable x_i as either being *left* (x_i = TRUE) or *right* (x_i = FALSE).

The clause part has $2m$ logical tracks, 2 for each clause. Precisely the trains corresponding to literals occurring in this clause have a position in the chain sequence turned on, namely in a way that the chain is placed on one of the two tracks. The first track of clause j stands for a literal making this clause true (contributing to set C_j^+), the second for one making it false (contributing to set C_j^-). From above it follows that there can be no gaps in the clause parts. We indicate the occurrences of literals in clauses by turning on the corresponding position in the chain sequence, as exemplified in Figure 5. The chains for each literal can be either left or right and therefore contribute either to C_j^+ or C_j^- for each clause j. By setting the capacity constraint to $2n + 2k - 2$ for each logical track in the clause part we enforce the not all equal constraint. This follows because this capacity limit is exceeded if and only if three literals contribute to the same of the sets C_j^+ and C_j^-. Therefore, under the assumptions above the transformed instance is a yes-instance for NAE3SAT if and only if there is a classification schedule that respects the given capacity bound.

It remains to specify how to enforce the two assumed properties. First, we want to replace the individual capacity constraints by a uniform one. To this end, we add one chain sequence of full length M. As every car is only allowed to be pulled once, the classification schedule for this chain sequence is unique. By adjusting the lengths of the chains of this chain sequence, the differences in the capacity constraints can be adjusted.

To enforce that every car is pulled at most once, we add one chain sequence with one chain of length 1 and one long chain. The length of the latter chain is exactly the excess capacity of the logical tracks w.r.t. all chain sequences constructed before. Now, if any car were pulled twice another car could not be pulled at all, which is impossible in a correct classification schedule. □

Single Outgoing Train The considerations so far depend on multiple outgoing trains. Here, we show the complexity result holds even for a single outgoing train.

Theorem 7. *For a single incoming and single outgoing train, with tracks of bounded capacity, the problem of finding an optimal classification schedule is NP-hard.*

Proof. We reduce the problem with multiple outgoing trains to this one. Let $T_1, \ldots, T_k$ denote the compact chain sequences of the k outgoing trains, and denote by $|T_i|$ the total number of cars in train i, not counting the first chain of length one that is not pulled because it goes directly to the outgoing train. Let h be the limit on the number of sorting steps, and C the capacity limit. Further, assume $\sum_i |T_i| = h \cdot C$ so that the cars must be pulled exactly once. This is the kind of instance described in the proof of Theorem 6.

We define an instance with a single outgoing train where the original cars are pulled precisely twice and there are additional long chains that are pulled precisely once. Let $M = \max |T_i| + 1$ so that all original chains are smaller than M. Define the new capacity to be $D = 2M$ and the new number of logical tracks $H = k + h$. Define further $d_i = D - |T_i|$ and $e = D - C$, and observe that $d_i > M$ and $e > M$. Now, the single outgoing train has the compact chain sequence

$$T = \langle T_1, d_1, T_2, d_2, \ldots, T_k, d_k, \underbrace{e, \ldots, e}_{\times h} \rangle$$

The enforced utilization of the tracks is schematically depicted in Figure 6.

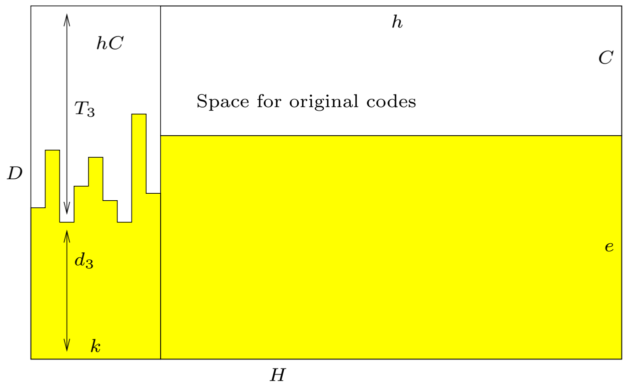

Fig. 6. Track utilization scheme for the transformation into a single outgoing train.

It remains to show that T can be sorted in H steps on a capacity D yard if and only if the T_i can be sorted simultaneously in h steps on a capacity C yard. Assume the latter, and let B_i be the bitstrings for train i (having a single 1 by assumption). We precede the bitstrings of B_i by 0s, only at position i we place a one. Further, the first chain gets bitstring 0, the chain d_i the bitstring with a single 1 at position i, and the chains of length e get the remaining bitstrings with a single 1, namely at positions $k+1, \ldots, H$. This set of bitstrings gives the correct order and uses capacity D on every position: On the first k positions because of $d_i + |T_i| = D$, and on the remaining positions because the original bitstrings used precisely capacity C. Hence, the train T can be sorted as claimed.

Now, conversely, assume that T can be sorted. Because $d_i > M$ and $e > M$, the minimal number of ones (car pulls) is given by a set of bitstrings that uses singleton bitstrings for the d_i and e chains and bitstrings with two 1s for the chains of the T_i. Such a set of bitstrings uses precisely $D \cdot H$ ones, and hence every correct set of bitstrings must have this structure. Because of the ordering of T, the bitstring of d_i must have the 1 at position i and the bitstrings of T_i must have their first 1 at position i. The singleton bitstrings for the e chains leave precisely a capacity C on the h tracks $k+1, \ldots, H$. Hence, the second 1s of the bitstrings for the T_i can be interpreted as bitstrings for the multiple train instance because they are all in the last h positions. They meet the length and capacity requirement and achieve the required ordering because the set of bitstrings was correct for the single train instance. $\square$

6.2 Approximation

In the previous section the train classification problem with restricted track capacities has been shown to be NP-hard. In the following a 2-approximation is derived, which basically consists of two consecutive steps: the procedure starts with a schedule not necessarily satisfying the capacity constraints (1) but the relaxed constraint (2) given below. This schedule is derived according to the proof of Lemma 6. Secondly, in this schedule every column violating (1) is distributed over several newly introduced columns which meet the capacity constraint as shown in Theorem 8 and Figure 7.

Consider the following constraint on the number of cars rolled in by a schedule B of length h, i.e., on the number of bits b_i^j in B with $b_i^j = 1$:

$$\sum_{i=1}^{h}\sum_{j=1}^{n} b_i^j \leq Ch \tag{2}$$

Every schedule B of length h that satisfies (1) also satisfies (2), whereas the converse implication does not hold in general.

Lemma 6. *Given a train T of length n, a minimum-length schedule satisfying (2) can be obtained in polynomial time in the size of the schedule encoding.*

Proof. Observe that there is no loss of generality in assuming that the schedule uses the same code for all cars of one chain: If this should not be the case for a chain, choose the code in the chain with the fewest 1s and use it for all cars of the chain. This schedule is still valid and does not use more roll-ins than the original one.

For a fixed number h, let B_h denote a valid schedule of length h that minimizes the number of cars rolled in. As detailed below, B_h can be obtained in polynomial time by a dynamic programming approach, which shows the statement of the lemma.

To describe the approach, it is useful to consider the following alternative description of a schedule: Double the code length and change a code $b_h \dots b_1$ into the code $b_h 1 b_{h-1} 1 \dots b_1 1$, i.e., on every other position there are only ones. The corresponding tracks, so called virtual tracks, contain all cars of the train, and the ordering of the cars summarizes the effect of the already performed shunting steps. More precisely, we see fewer chains, for example with the complete binary code, every shunting steps reduces the number of chains by a factor of two. In general, the difference in the sequence achieved by one shunting step is that some chains get united, namely consecutive pairs of chains. The chain $c'c''$ is created from the higher numbered chain c'' that has a 1 and the chain c' tat has a 0 at the position. Because the set of cars of disjoint resulting chains are disjoint, there is no interaction between creating c' and c''.

The dynamic program has a table of the form $D(i, c)$, where c is any intermediate chain, i.e., an interval of cars in the outgoing train, and $D(i, c)$ gives the fewest possible roll-ins to create c with i shunting steps (on the ith virtual track). Clearly, $D(0, c) = 0$ precisely for the original chains; for all other chains $D(0, c') = \infty$. For $i > 0$, the above discussion justifies the formula

$$D(i,c) = \min_{c'c''=c} \left(D(i-1,c') + D(i-1,c'') + |c''|\right)$$

In this formula, $c'c''$ denotes the concatenation of c' and c'' and $|c''|$ the length of c'', which gives precisely the number of roll-ins necessary to place c'' (with 1 at the position) behind c' (with 0 at the position). Note that we allow c'' to be the empty chain, i.e., it does not get more expensive to create a chain at a later stage.

The interesting entry of the table is $D(h, 1\dots n)$ which should be compared with the constraint (2). The schedule can be reconstructed in the usual manner by keeping track of the split that achieved the minimum. The validity and total number of roll-ins of the schedule follows from the chains appearing on the virtual tracks as claimed by the dynamic programming table.

A single entry $D(i,c)$ can be calculated in time $\mathcal{O}(|c|)$, so the total running time for the dynamic programming part of the algorithm is $\mathcal{O}(hn^3)$ since the number of chains, i.e., the width of the table, is given by $\binom{n}{2} \in \mathcal{O}(n^2)$. The reconstruction time for the schedule B is linear in the size of B, which is $\mathcal{O}(hn)$. □

Increasing the length of a valid schedule allows reducing the number of cars rolled in. Thus, if B' is the shortest valid schedule that satisfies (2) and h' denotes its length, the length h^* of the optimal schedule B^* meeting (1) satisfies $h^* \geq h'$. 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.

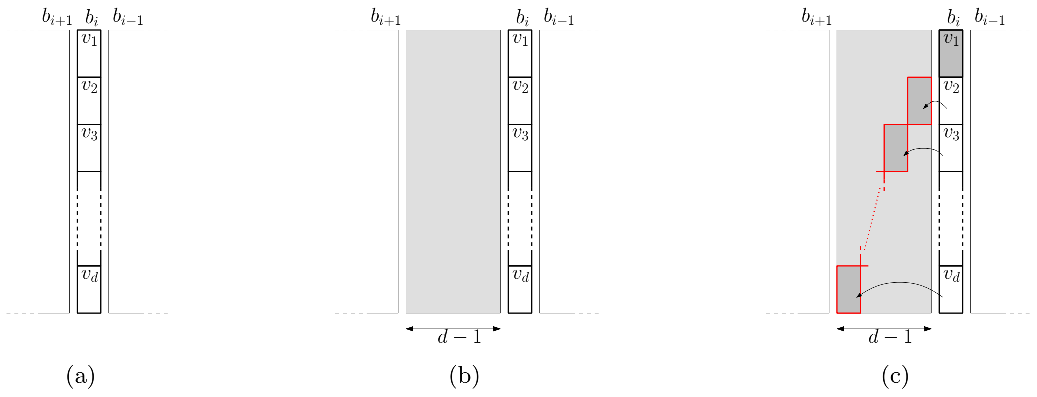

Fig. 7. One iteration step of the approximation algorithm for bounded capacities, applied to a column b_i that violates the capacity constraints (1).

Theorem 8. *For any instance of the train classification problem with restricted track capacities that has an optimal solution of length h^*, a valid schedule of length of at most $2h^*$ can be obtained in polynomial time.*

Proof. Let B be a schedule of minimum length h that is valid w.r.t. the unrestricted problem and further satisfies (2). (B can be obtained according to Lemma 6.) This schedule is transformed iteratively, performing one iteration step for every column of B that violates (1). The steps can be performed in an arbitrary order. Figure 7 depicts a single iteration step.

For any column vector b_i of B, let the *load* ℓ_i of b_i be defined by $\ell_i := \sum_{j=1}^{n} b_i^j$. Now, let b_i be a column vector with $\ell_i > C$. Define $d := \lceil \frac{\ell_i}{C} \rceil$ and divide b_i into d consecutive subvectors $v_1, \ldots, v_d$ such that each v_k with $k < d$ has a load of exactly C and the load of v_d is at most C (Figure 7(a)). Then, shift all columns b^j with $j > i$ exactly $d-1$ positions to the left by inserting $d-1$ additional all-zero columns (Figure 7(b)). Finally, every v_k is moved to the left by $k-1$ positions, and the corresponding entries in b_i are set to zero, which finishes the iteration step (Figure 7(c)).

After each step, the respective fixed column satisfies (1), which also holds for every newly introduced column, so the resulting schedule satisfies (1).

For every column in B, this procedure yields at most one "underfull" column (load less than C), and the total number of "tight" columns (load equal to C) will not exceed h, so the resulting length is at most $2h$. Since $h \leq h^*$, where h^* denotes the length of an optimal schedule satisfying (1), the resulting valid schedule has a length of at most $2h^*$. □

The algorithm of this section can easily extended to multiple outgoing trains $S_1, \ldots, S_m$: in the proof of Lemma 6, one table must be maintained *for each* outgoing train S_k, $k = 1, \ldots, m$. In each step i, one row $D(i, \cdot)$ is constructed *in each table*, which is repeated until $\sum_{k=1}^{m} D(h, S_k) \leq hC$ after some step h. A single table entry is still derived only from entries in the same table, and also the reconstruction can be done independently for every train. Finally, the proof of Theroem 8 can be applied to the resulting schedule without any changes, just as for a single outgoing train.

6.3 Chains of Unit Length

As shown in Section 6.1, the train classification problem with tracks of bounded length is NP-hard. In the following, however, we show that, if all chains consist of single cars, optimal schedules can be constructed efficiently (in the size of the resulting encoding). These schedules also work for situations with longer chains, the schedules are just not optimal any more. This corresponds to a planning situation where the ordering of the cars in the incoming train is not known and the proposed schedule must work for any ordering that the train actually has.

Once again, classification schedules that are valid for tracks of bounded capacity C have binary encodings B that satisfy constraint (1). In other words, for every column the total number of 1s is bounded by C.

Algorithm to produce an optimal set of bitstrings Note that the maximal set of bitstrings for h positions uses 2^{h-1} ones at each position. If this exceeds the capacity (track length) only by one, we can omit the single bitstring that has a 1 in every position. In the other extreme, if the capacity is one, the largest set of bitstrings consists of the h bitstrings with a single 1. We analyze this situation more deeply in the following example. Assume that for the given capacity c, one can use all bitstrings with a single 1 (singletons), all bitstrings with two 1s (pairs), and as many bitstrings with three 1s (triples) as possible. Assume this can be done in a way such that all but at most two positions are at the capacity limit c, and the remaining (≤ 2) positions have $c-1$ 1s. Then, this set of bitstrings is clearly optimal, since any other way of choosing this many bitstrings will use at least the same number of 1s in total. Using all bitstrings with a certain number of 1s per bitstring will use the same capacity on all tracks.

The following lemma and its constructive proof show how to produce optimal sets of bitstrings for all capacities. In our example from above, let c be the capacity left after all singletons and pairs are assigned. In the above example, we can choose $i = 3$, $n = \lfloor \frac{c \cdot h}{3} \rfloor$ in the following lemma to complete the set of bitstrings. Note that the order of the positions is arbitrary.

Lemma 7. *Let h, n, i be positive integers with $h \geq i$, and $n \leq \binom{h}{i}$. Then, there is a set of n bitstrings of length h with precisely i 1s in each bitstring, such that the total number of 1s at two positions differs by at most one. That is, there are numbers $\overleftarrow{h} + \overrightarrow{h} = h$ and $c = \lceil \frac{ni}{h} \rceil$ such that in $\overleftarrow{h}$ positions the total number of 1s is c, and in $\overrightarrow{h}$ positions the total number of 1s is $c-1$.*

Proof. The proof is by induction on h. The base case is $h = i$. Then $n \leq \binom{h}{i}$ yields $n = 1$. Now the bitstring consisting of $i = h$ 1s (and no 0s) shows the statement of the lemma.

Now assume $h > i$ and inductively assume that the lemma holds for all i and all $h' < h$. In one inductive step a set S_1 of c bitstrings with 1s in the first position is created. In the example of Figure 8, $S_1 = \{a, b, c, d\}$. The remaining $(i-1)|S_1|$ 1s of the bitstrings in S_1 are equally distributed over all remaining positions by an inductive application of the lemma. In the example, this is the pseudo-box in the lower right corner. Again inductively, the remaining bitstrings S_2 have a 0 in the first position; in the example $S_2 = \{e, f, g, h\}$.

Because the positions are arbitrary, we can arrange the inductive results in a way that distributes the used capacity evenly. More precisely, we can assume that the bitstrings for S_1 have all higher capacity positions to the right, whereas the bitstrings for S_2 have the higher capacity positions to the left. Now all positions but a middle segment have the same capacity (one higher plus one lower). The middle segment deviates from this capacity by at most one.

To complete the proof of the lemma, we only have to show that for both inductive applications the two assumed inequalities are met. We define some abbreviations with the following naming conventions: Subscripted values are associated with the respective invocation of the lemma. Hence, the first invocation of the lemma is with $h_1 = h - 1$ as

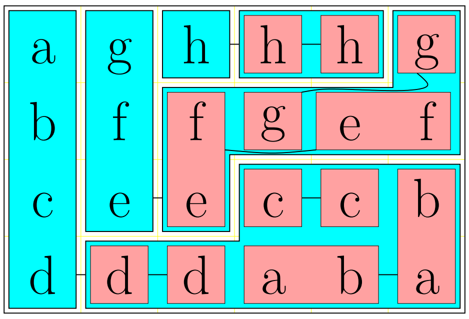

Fig. 8. Illustration of the recursive bitstring generation with three 1s per bitstring, 6 tracks of length 4, yielding 8 bitstrings. The recursion uses all remaining 1s of a position (the vertical box), and recursively adds the remaining positions of the bitstrings (the flat pseudo-box). Importantly, the flat pseudo-boxes are chosen in a way that the remaining space is almost equal in the different positions.

the bitstring length, $i_1 = i - 1$ the number of 1s per bitstring, and $n_1 = c$ the number of bitstrings, resulting in S_1, where the positions with more 1s are higher numbered (to the right). We invoke it another time with $h_2 = h - 1$ as the bitstring length, $i_2 = i$ 1s per bitstring, and $n_2 = n - c$ bitstrings, resulting in the set of bitstrings S_2, where the positions with more 1s are lower numbered (to the left).

Here $h_1 \geq i_1$ and $h_2 \geq i_2$ hold because we are not in the base case. To show the inequalities involving the binomial coefficient, it is convenient to combine the two "fill-heights" (or deviation from the maximal possible fill height), c and $c-1$, into an averaged value $\hat{c} = \frac{ni}{h}$. We mark all such average values by a hat. The average height resulting from S_1 on the remaining positions is necessarily $\hat{c}_1 = c(i-1)/(h-1)$.

The bound on n implies $\hat{c} \leq \binom{h-1}{i-1}$ (an alternative way to describe the assumption of the lemma) because $ni \leq i \cdot \binom{h}{i} = \frac{h}{i} \cdot i \cdot \binom{h-1}{i-1}$ which yields $\hat{c} = \frac{ni}{h} \leq \binom{h-1}{i-1}$. This gives the required $n_1 = c \leq \binom{h-1}{i-1}$ by integrality of the binomial coefficient. Hence, the assumptions for the first recursive invocation of the lemma are met.

Now, let us define $e = \binom{h-1}{i-1} - c$, the deviation from the maximal possible capacity. For the recursive applications of the lemma we are only interested in the last $h-1$ positions. We define the average total height for both recursive applications for these positions only: $\hat{c}_{1+2} = \frac{\overleftarrow{h} - 1}{h} c + \frac{\overrightarrow{h}}{h}(c-1) = \frac{ni-c}{h-1} \leq c$, and the corresponding $\hat{e}_{1+2} = \binom{h-1}{i-1} - \hat{c}_{1+2}$. Observe that $e \leq \hat{e}_{1+2} (\leq e+1)$. For the second invocation of the lemma we should check $\hat{c}_2 \leq \binom{h_2-1}{i_2-1} = \binom{h-2}{i-1}$, where $\hat{c}_2 = \hat{c}_{1+2} - \hat{c}_1$. Observe $\frac{h-1}{i-1}\hat{c}_1 = c = \binom{h-1}{i-1} - e$ such that we can express $\hat{c}_1 = \frac{i-1}{h-1}\binom{h-1}{i-1} - \frac{i-1}{h-1}e = \binom{h-2}{i-2} - \frac{i-1}{h-1}e$. So, $\hat{c}_2 = \hat{c}_{1+2} - \hat{c}_1 = \binom{h-1}{i-1} - \hat{e}_{1+2} - \binom{h-2}{i-2} + \frac{i-1}{h-1}e = \binom{h-2}{i-1} + \frac{i-1}{h-1}e - \hat{e}_{1+2} \leq \binom{h-2}{i-1}$, where the last inequality relies upon $h \geq i$ making the coefficient of e smaller than one, hence the statement follows by the already observed $e \leq \hat{e}_{1+2}$. $\square$

The recursive procedure implied by the proof of Lemma 7 is efficient: The initial task of using $h \cdot i$ 1s in bitstrings is split into two subtasks. Hence, the number of leaves in the recursion tree is $O(h \cdot i)$, and the total running time is polynomial.

Theorem 9. *For positive integers h and C a set B of bitstrings of length h that obey capacity constraint (1) of maximal cardinality can be constructed in time polynomial in $|B|$.*

Proof. For some i, it is possible to use all bitstrings of length h with precisely j 1s for $j = 1, \ldots, i-1$. The remaining bitstrings are constructed according to Lemma 7 (actually its constructive proof) by taking the largest n (binary search) for which the resulting set of bitstrings leaves at most $i-1$ positions at capacity $C-1$, and all other positions with capacity C. The binary search can be avoided by formulating two procedures, one that takes a parameter n, and another one that produces as many bitstrings as possible given the constraints. □

7 Secondary Objectives

The quality of a classification schedule is mainly determined by the time it takes to execute the complete schedule. So far, we have estimated this time by the number of required sorting steps. On the other hand, it is also practice to consider the total number of cars rolled in as the main objective. (At some classification yards, this objective is used for pricing classification requests of customers.) Clearly, both objectives contribute to the total required time, and, in [23], their relation is summarized as follows: if the constants c_{pull}, c_{roll}, and c_{push} denote the time required for pulling out one track, the time the last car needs to roll from the hump to a track, and the time for decoupling and pushing one car, respectively, then a classification process of h steps and a total of r cars rolled in roughly requires a time of $h(c_{\text{pull}} + c_{\text{roll}}) + rc_{\text{push}}$. As mentioned before, the term $h(c_{\text{pull}} + c_{\text{roll}})$ dominates rc_{push}, hence the choice of h as our primary objective. A detailed example supporting this idea with actual numbers for the constants and experimental results for h and r is also given in [23].

Still, the total number of cars that are pushed over the hump and rolled into the classification tracks also influences the total execution time. This secondary objective, the number of roll-ins, equals the number of 1's in an encoding. It can be incorporated into the objective function in various ways. First, one could ask for the optimal solution with respect to sorting steps that minimizes the number r of car roll-ins. We call such a solution a *slim h-optimal* solution. A more general approach is to charge a cost of α for a pull-out of a train (e.g., $\alpha = c_{\text{pull}} + c_{\text{roll}}$ from above) and β for a roll-in of a single car (e.g., $\beta = c_{\text{push}}$). We call a solution that is optimal w.r.t. this objective *(h, r)-optimal*. In the following, we briefly discuss the consequences of the two new objective functions.

Until now we could safely assume without loss of generality that in each step a *complete* track is pulled out. This situation is also common practice in real-world classification yards. The following lemma shows that, in case of unrestricted capacity and width, it is not advantageous to only partially pull-out a track, i.e., to leave some cars on the track and to pull-out the rest.

Lemma 8 (partial pull-outs). *If we assume an unbounded number of tracks in a model with or without track capacity, there is both a slim h-optimal and a (h, r)-optimal solution without partial pull-outs. For the case of bounded width W it can be necessary to use partial pull-outs to get a solution that is slim h- or (h, r)-optimal.*

Proof. For the first statement, observe that any solution in which a track is partially pulled out can be transformed into a solution on one more track in which the two parts are rolled into different tracks. For the second statement, consider an example with $W = 2$ and $n = 4$ and assume unit chain length for simplicity. From Section 5 we know that since n is between the fourth and the fifth Fibonacci number any feasible encoding scheme needs at least five bit bitstrings that start and end with a one. A solution with minimum number of ones without leading and trailing ones is $010, 011, 101, 110$. Without partial pull-outs the cost of this solution is 7 plus the initial and final roll-in and cannot be improved. This solution, however, can keep the chain of bitstring 101 in its track, as it is the bottommost chain in the step where it is rolled into the same track again. Thus one roll-in can be saved by a partial pull-out, the slim h-optimal solution must use pull-outs, analogously for (h, r)-optimal solutions. □

We already know that it is NP-hard to find the optimal classification schedule for capacity-bounded tracks. Therefore, we cannot hope for a polynomial algorithm for the new objectives in this case. The basic version of the problem remains polynomial, however, as shown in the following lemma.

Lemma 9. *A slim h-optimal solution can be computed in polynomial time for the case of unbounded number of tracks and unbounded track capacity.*

Proof. For a given incoming train we can compute all chain lengths. The remaining problem is to select n bitstrings from the $2^{\lceil \log n \rceil}$ candidate bitstrings that make the solution slim. The best subset of the n bitstrings has a minimum scalar product of the vector of chain lengths and the vector of number of ones in the selected bitstrings. This subset can be found in the obvious way by a dynamic program[5] that computes a table of numbers $C(i, j)$ that represent the minimum cost of a solution for the first i chains that uses the first j bitstrings. □

The partial pull-outs do not invalidate the encoding scheme. For a given encoding it is possible to read off the number of roll-ins with partial pull-outs. This is stated in the following observation that directly follows from Lemma 1.

[5] Unlike stated in the conference version of this article [20], the set of bitstrings with the minimum number of ones is not necessarily an optimal solution. This statement only holds for chains of unit length.

Observation 1 *A car τ_j of bitstring $c = X \overbrace{1}^{pos\ i} Y \in \{0,1\}^h$ in an encoding for train T is the front element in step i if for all $\tau_k \in T \setminus \tau_j$ of bitstring $X_k 1 Y'_k$ it holds that $Y < Y_k \vee (Y = Y_k \wedge j < k)$.*

We have focussed on slim h-optimal solutions here. The extension to (h, r)-optimal solutions is straightforward in the sense that one can always search through the possible values for h with $\lceil \log n \rceil \leq h \leq n$.

8 Concluding Remarks

We have developed an efficient encoding of freight train classification schedules that allows us to present, analyze, and develop train classification methods for real-world classification yards. This surprisingly simple yet powerful encoding can be used to analyze the efficiency of commonly used multistage methods. We proved the optimality of the simultaneous variant geometric sorting in terms of sorting steps, considering presorted input.

Practical Considerations Dealing with single and multiple incoming and outgoing trains, with yards of restricted capacity and number of tracks, and with presorted input, the results of this paper cover the core of the sorting task at real-world classification yards. We have shown that the constraint on the capacities leads to an algorithmically more difficult problem than the constraint on the number of tracks. Fortunately, in practice, the number of classification tracks presents the more restrictive constraint in modern classification yards.

Railway practitioners may notice that some aspects and exceptions of real-world classification tasks are not covered here. Many of these aspects, however, can be incorporated in our encoding scheme by a simple restriction on the bitstrings in an encoding. In [23], we present an integer programming approach for the train classification problem in which we indeed incorporate the following three restrictions:

First, in many classification yards, multistage sorting is performed for local freight trains while the rest of the traffic is sorted by single-stage sorting. Primary sorting (as the first step of multistage sorting) and single-stage sorting are performed at the same time here. Secondary sorting is usually started after all trains from single-stage sorting have left the yard, so more tracks are available for secondary sorting. In [23], we incorporate this requirement in a way similar to the considerations of Section 5.

Second, there are classification yards featuring two parallel humps, such as the yard of Lausanne Triage considered in [23]. In this case, two independent multistage sorting processes can be performed: each hump is assigned a (disjoint) fixed set of classification tracks and a subset of the outgoing trains. Incoming cars are assigned to the humps according to their outgoing train and remain in the system of that hump during all primary and secondary sorting.

Third, some outgoing trains may have to leave before the complete multistage process finishes. As each sorting step roughly requires a constant amount of time, the required departure times can be assigned to the sorting steps. Hence, for every car, the latest possible step can be calculated after which this car must be on the train formation track. This is the last restriction incorporated in [23], and further practical restrictions may be possible to incorporate too.

Furthermore, there are yards with a secondary hump at their opposite ends like in Figure 2(c). As secondary sorting can be performed via this hump without blocking the primary hump, it can be started without retarding the single-stage sorting process. As mentioned in Section 1, secondary humps are not always used due to cost and operational reasons.

Future Work 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.

Furthermore, it would be interesting to apply some of the theoretical results of this article to improve the actual classification procedures applied in real-world classification yards. To this end, we are currently conducting a computer simulation for the Swiss classification yard of Lausanne Triage with real-world traffic data. In [23], using the integer programming model mentioned above, we successfully derived a schedule for the traffic data of a whole day that is one step shorter and requires one track less than the currently applied method. We are going to verify the applicability of this schedule in the mentioned computer simulation, the results of which we are going to publish in a follow-up paper.

Finally, much time is lost in the classification process by the practice of starting secondary sorting not before all single-stage sorting is finished. This may be overcome by performing both methods at the same time, which requires considering the arrival and departure times of trains. As mentioned above, a first step to this direction is made in [23] with incorporating train departures for the multistage method. 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 reduce dwell times, improve departure times, and increase the traffic throughput of classification yards.

Acknowledgments

We would like to thank M. Mihalak for his valuable comments on a previous version of this article. Thank you very much also to N. S. Narayanaswamy for the inspiring discussions on the relation of OPT-PERM to minimum feedback arc set problems. Last but not least, we would like to thank the anonymous reviewers for their detailed and useful reviews.

References

1. R.K. Ahuja, K.C. Jha, and J. Liu, Solving real-life railroad blocking problems, Interfaces 37 (2007), 404–419.
2. O. Baumann, Die Planung der Simultanformation von Nahgüterzügen für den Rangierbahnhof Zürich-Limmattal, ETR RT 19 (1959), 25–35.
3. B.C.M. Boot, Zugbildung in Holland, ETR RT 17 (1957), 28–32.
4. M. Campetella, G. Lulli, U. Pietropaoli, and N. Ricciardi, Freight service design for the italian railways company, ATMOS 2006 - 6th Workshop Algorithmic Methods Models Optim Railways, IBFI, Schloss Dagstuhl, Germany, 2006, pp. , <http://drops.dagstuhl.de/opus/volltexte/2006/685>.
5. J.F. Cordeau, P. Toth, and D. Vigo, A survey of optimization models for train routing and scheduling, Transportation Sci 32 (1998), 380–404.
6. C.F. Daganzo, Static blocking at railyards: Sorting implications and track requirements, Transportation Sci 20 (1986), 189–199.
7. C.F. Daganzo, Dynamic blocking for railyards: Part I. homogeneous traffic, Transportation Res 21B (1987), 1–27.
8. C.F. Daganzo, Dynamic blocking for railyards: Part II. heterogeneous traffic, Transportation Res 21B (1987), 29–40.
9. C.F. Daganzo, R.G. Dowling, and R.W. Hall, Railroad classification yard throughput: The case of multistage triangular sorting, Transportation Res, Part A 17 (1983), 95–106.
10. E. Dahlhaus, P. Horák, M. Miller, and J.F. Ryan, The train marshalling problem, Discr Appl Math 103 (2000), 41–54.
11. E. Dahlhaus, F. Manne, M. Miller, and J. Ryan, Algorithms for combinatorial problems related to train marshalling, Proc. 11th Australasian Workshop Combinatorial Algorithms (AWOCA-00), 2000, pp. 7–16.
12. G. Di Stefano, J. Maue, M. Modelski, A. Navarra, M. Nunkesser, and J. van den Broek, Models for rearranging train cars, Technical report TR-0089, ARRIVAL, 2007.
13. K. Endmann, Untersuchungen über die Simultanzugbildung bei der Deutschen Bundesbahn, Bundesbahn 37 (1963), 593–600.
14. G. Even, J. Naor, B. Schieber, and M. Sudan, Approximating minimum feedback sets and multi-cuts in directed graphs, Proc 4th Int Conference Integer Program Combinatorial Optim, Vol. 920 of *LNCS*, 1995, pp. 14–28.
15. P. Festa, P.M. Pardalos, and M.G.C. Resende, *Handbook of Combinatorial Optimization* Vol. 4 chapter Feedback set problems, pp. 209–258, Kluwer Academic Publishers 1999.
16. H. Flandorffer, Vereinfachte Güterzugbildung, ETR RT 13 (1953), 114–118.
17. M.R. Garey and D.S. Johnson, Computers and intractability, Freeman, 1979.
18. R.S. Hansmann and U.T. Zimmermann, "Optimal sorting of rolling stock at hump yards," Mathematics - key technology for the future: Joint projects between universities and industry, Springer, 2007, pp. 189–203.
19. H.P. Holliger, Rangierbahnhof Limmattal, webpage. Personal communication.
20. R. Jacob, P. Marton, J. Maue, and M. Nunkesser, Multistage methods for freight train classification, Proc. 7th Workshop Algorithmic Methods Models Optim Railways (ATMOS-07), Wadern, Germany, IBFI Schloss Dagstuhl, 2007, pp. 158–174.
21. K. Krell, Grundgedanken des Simultanverfahrens, ETR RT 22 (1962), 15–23.

22. K. Krell, Ein Beitrag zur gemeinsamen Nutzung von Nahgüterzügen, ETR RT 23 (1963), 16–25.
23. J. Maue and M. Nunkesser, Evaluation of computational methods for freight train classification schedules, Technical report TR-0184, ARRIVAL, 2009.
24. A. Newman, The maximum acyclic subgraph problem and degree-3 graphs, Proc 4th Int Workshop Approximation Algorithms Combinatorial Optim Prob, APPROX, Vol. 2129 of *LNCS*, 2001, pp. 147–158.
25. K.J. Pentinga, Teaching simultaneous marshalling, Railway Gazette (1959), 590–593.
26. M.W. Siddiqee, Investigation of sorting and train formation schemes for a railroad hump yard, Proc. 5th Int. Symp Theory Traffic Flow Transportation, 1972, pp. 377–387.