Robust Algorithms for Sorting Railway Cars

Christina Büsing[1] and Jens Maue[2]

[1] Institut für Mathematik, Technische Universität Berlin, Germany
cbuesing@math.tu-berlin.de
[2] Institute of Theoretical Computer Science, ETH Zürich, Switzerland
jens.maue@inf.ethz.ch

Abstract. We consider a sorting problem from railway optimization called train classification: incoming trains are split up into their single cars and reassembled to form new outgoing trains. Trains are subject to delay, which may turn a prepared sorting schedule infeasible for the disturbed situation. The classification methods applied today deal with this issue by completely disregarding the input order of cars, which provides robustness against any amount of disturbance but also wastes the potential contained in the a priori knowledge about the input.

We introduce a new method that provides a feasible sorting schedule for the expected input and allows to flexibly insert additional sorting steps if the schedule has become infeasible after revealing the disturbed input. By excluding disruptions that almost never occur from our consideration, we obtain a classification process that is quicker than the current railway practice but still provides robustness against realistic delays. In fact, our algorithm allows flexibly trading off fast classification against high degrees of robustness depending on the respective need. We further explore this flexibility in experiments on real-world traffic data, underlining our algorithm improves on the methods currently applied in practice.

1 Introduction

An essential process in railway optimization is *train classification*, which refers to the rearrangement of cars to form new trains. With increasing world-wide freight traffic, operating freight trains efficiently becomes more and more important, and reducing the dwell time of cars in railway yards is one of the key factors to improve freight service profitability.

General Classification Process. Exclusively for the purpose of train classification, there are installations of railway tracks and switches called *classification yards* (see Fig. 1). Such a yard features a *hump track* on which inbound trains arrive and their cars are decoupled to be pushed over a sloping ramp called *hump* at the end of the hump track. Hence, the cars accelerate by gravity and roll through a tree of switches by which each car can be individually guided to some *classification track*. This is called a *roll-in* operation. In a *pull-out* operation an engine pulls all the cars on some classification track back to the hump track in order to perform a further roll-in. A pair of pull-out and roll-in operations is called a *(sorting) step*, and an initial roll-in followed by a sequence of h sorting

M. de Berg and U. Meyer (Eds.): ESA 2010, Part I, LNCS 6346, pp. 350–361, 2010.

steps is called a *classification schedule* of *length* h. The number of steps h essentially determines the time required to conclude the sorting procedure. There are ℓ inbound trains that, concatenated in the order they arrive at the yard, form the *inbound train sequence*. Moreover, there are order specifications for the m *outbound trains*, and a classification schedule is called *feasible* if its application to the inbound train sequence yields the correctly ordered outbound trains, each on a separate classification track.

Robust Train Classification. Often the inbound trains are subject to delay, so we might be faced with an unexpected inbound order of trains. In our model all disturbances, i.e. every combination of number of delayed trains and amount of delay for each train, that are to be covered are given by a *set of scenarios* $\mathcal{S}$. In this set, each scenario $S \in \mathcal{S}$ defines a permutation of the inbound train sequence called *modified instance*. A schedule for the original instance is called a *first-stage solution*, and it may be infeasible for the modified instance corresponding to some scenario. In response to disturbed input, we are prepared to insert up to k additional sorting steps after the pth step of the first-stage solution, providing a *recovered solution*. A first-stage solution for which, for every scenario $S \in \mathcal{S}$, there is a recovered solution that is feasible w.r.t. S is called *recovery robust*. Given a sequence of ℓ inbound trains, m order specifications of outbound trains, and a set of scenarios $\mathcal{S}$, the recovery-robust train classification problem is to find a recovery robust, feasible first-stage solution of minimum length.

Related Work. There are many publications in the field of railway engineering that describe different train classification methods, e.g. [7,16,13,17,5]. These methods are *strictly robust*, i.e. robust w.r.t. any set of scenarios, since they apply a predefined classification schedule that is independent of the order of railway cars entering the classification process. The method of *geometric sorting* (see [7,16,13,17]) minimizes the number of sorting steps for a worst case (or unknown) input order, which is proved in [11]. The still most-commonly used method in practice is *triangular sorting* [7,16,13,17,5], which is optimal for restricting the number of roll-ins per car to three for unknown input order [11]. However, neither method exploits the situation of a partially ordered input sequence, so they apply more sorting steps than necessary in general.

This issue was explored in [11], which develops a classification method that minimizes the number of sorting steps based on complete knowledge of the input data. Moreover, for the problem variant of classification tracks of bounded length, [11] shows that minimizing the number of sorting steps is an NP-hard problem. A 2-approximation for the same setting is derived in [12], several improvements of which are experimentally evaluated in [10] and compared to an exact integer programming approach, which was earlier introduced in [15]. A related algorithmic sorting problem is considered by Dahlhaus et al. [6]. Recent overviews of train classification can be found in [9] and [8].

Since changes during the process of scheduling are time consuming, a certain amount of robustness is crucial for classification methods to work in practice. Providing strict robustness, however, wastes a lot of potential to disruption scenarios that almost never occur in practice. As described above w.r.t. train

classification, this dilemma is tackled by the concept of recoverable robustness [14] by regarding realistic scenarios of delay and providing optimal robust solutions w.r.t. a limited amount of recovery in case of disturbance. This concept is applied to several railway-related optimization problems such as rolling stock scheduling [1] or timetabling [3,4]. A first and—to the best of our knowledge—only attempt to study this method for train classification is made by Cicerone et. al [2] for a single inbound and outbound train. (Their results are summarized in [3].) Besides the situations of strict robustness and complete recomputation from scratch, which are more of theoretical interest, they consider a recovery action that allows completely changing the classification instruction for one set of cars that have the same instruction. The most relevant scenarios in [2] are one additional car in the input and one car occurring at a different position than expected. The latter corresponds to our problem setting for the special case of trains consisting of single cars with a delay scenario of up to one train. We generalize this setting to scenarios with more delays (mainly Sect. 4) and the problem setting with complex trains. Besides, [2] deals with the scenario of a single classification track becoming unavailable before the classification starts. In this paper we focus on the most relevant reason for disruptions, which are delayed trains.

Our Contribution. For the mentioned recovery action of adding up to k sorting steps after an offset of p steps, we first introduce a generic algorithm in Sect. 3. We prove that, for every constant $k \geq 1$, finding a robust schedule of minimum length is an **NP**-complete problem for general sets of scenarios. For the practically relevant scenario of delaying up to j trains by an arbitrary amount each, the problem can be solved in polynomial time (see Sect. 4). Furthermore, we evaluate our new algorithm on real-world traffic data for various parameter values k, p, and j. It turns out that, on the one hand, our algorithm yields very short schedules while providing a fair degree of robustness. On the other hand, it is capable of providing highly robust schedules that still improve on the current classification practice, emphasizing the flexibility of our approach to modulate between these conflicting objectives.

2 Encoding Classification Schedules

In addition to the concepts of Sect. 1, we introduce some futher notation required for representing and deriving classification schedules.

Terminology and Notation. Corresponding to the notation of [11], we represent every car τ by a positive integer $\tau \in \mathbb{N}$ and a train T by a sequence of cars $T = (\tau_1, \ldots, \tau_k)$, where k is called the *length* of T. There are ℓ *inbound* trains $T_1, \ldots, T_\ell$, whose concatenation we assume to be a permutation of $(1, \ldots, n)$, and n is called the *volume* of cars. There are m *outbound* trains of respective length n_i, $i = 1, \ldots, m$, and we assume the specification of the first outbound train is $(1, \ldots, n_1)$, the second $(n_1{+}1, \ldots, n_1{+}n_2)$, etc. In contrast to the expected order of inbound trains, there is no order implied for the outbound trains.

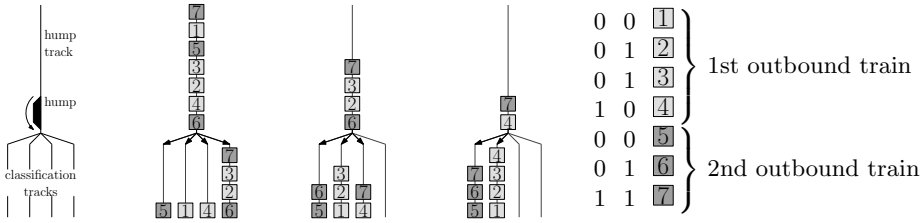

Fig. 1. Classification yard (left). Classification process for $h=2$, $n=7$, and $m=2$: initial roll-in (2nd picture), first step (3rd), second step (4th); in both steps, the *rightmost* occupied track is pulled out. Corresponding schedule encoding (right).

Regardless of which are their outbound trains, all cars are sorted simultaneously on the same set of classification tracks, called the *sorting tracks*. Their lengths and available number are unrestricted, and the number actually used corresponds to the number of sorting steps, where the track pulled in the kth step is referred to by θ_k, $k = 0, \ldots, h-1$. The cars are finally collected on a separate track for each outbound train, which are called *destination tracks*.

Schedule Representation. We will refer to the binary representation of a decimal integer $j \geq 0$ by $[j]_2$. Given any bitstring $b = b_{h-1} \ldots b_0$ of length h, let $\mathrm{num}(b)$ denote the integer number represented by b, i.e., $\mathrm{num}(b) = \sum_{i=0}^{h-1} 2^i b_i$. For two bitstrings b_1, b_2 we define $b_1 < b_2$ iff $\mathrm{num}(b_1) < \mathrm{num}(b_2)$. We represent classification schedules of length h by assignments of cars to bitstrings of length h [11]: $b^j = b^j_{h-1} \ldots b^j_0$ encodes the journey of the jth car with $b^j_k = 1$ iff it visits θ_k pulled out in the kth step. After such a pull-out, the car is sent to θ_ℓ with $\ell = \min\{i | k < i < h,\ b^j_i = 1\}$; if $b^j_i = 0$ for all $i > k$, it goes to the destination track of its outbound train. The n bitstrings $b^1, \ldots, b^n$ form an $(n \times h)$-matrix, and $b_0, \ldots, b_{h-1}$ denote its columns *from "right" to "left"*.

In order to derive a feasible schedule B of length h, two cars τ and $\tau{+}1$ of the same outgoing train must be assigned bitstrings $b^\tau \leq b^{\tau+1}$. If these cars occur in reversed order in the inbound sequence, we require $b^\tau < b^{\tau+1}$; then, the pair $\beta = (\tau, \tau{+}1)$ is called a *break*. If b^τ and $b^{\tau+1}$ occur in different outbound trains, there is no constraint between the two cars. As a result the length of a classification schedule depends on the maximum number of breaks of all outbound trains [12]. Figure 1 shows an example classification process and the corresponding encoding with four steps and tracks, where the rightmost track presents θ_0 in the notation above: cars 1 and 2 arrive in reversed order, so $b_1 < b_2$, whereas cars 2 and 3 arrive in correct order and have the same bitstring. Note that $b^7 > b^6$ is fine though cars 6 and 7 arrive in correct order, and there is no constraint between b_4 and b_5 since the fourth and fifth car belong to different outbound trains.

3 Recovery through Additional Sorting Steps

In this section we investigate the recovery strategy of inserting a limited number of additional sorting steps to a first-stage schedule when a scenario occurs.

Further Notation. A pair of consecutive cars $\beta = (\tau, \tau+1)$ is called *original break* if β is a break for the expected order of inbound trains. Given some $S \in \mathcal{S}$, we call β *induced by* S if β is a break in the modified instance corresponding to S. If β is not an original break but induced by any $S \in \mathcal{S}$, β is called a *potential break*. W.l.o.g., we assume that every pair $\beta = (\tau, \tau+1)$, $\tau \in \{1, \ldots, n-1\}$, of successive cars is either an original or a potential break: for any problem instance with β not being a break, car $\tau+1$ can be ignored while deriving a schedule and assigned the same bitstring as τ in the final solution. For any first-stage solution B, a break $\beta = (\tau, \tau+1)$ is called *unresolved* w.r.t. S if β is induced by S and $b^\tau = b^{\tau+1}$. For any scenario $S \in \mathcal{S}$, X^S denotes the set of potential breaks induced by S. Note that this set X^S is uniquely defined for every scenario $S \in \mathcal{S}$, but there may be different scenarios $S \neq S'$ with $X^S = X^{S'}$. We will repeatedly regard sets of potential breaks without considering the actual underlying scenario. In particular, we will often describe sets of scenarios (e.g. as a parameter in problem definitions) implicitly by providing the set of induced breaks of every scenario. Let $T_1, \ldots, T_\ell$ be a sequence of ℓ inbound trains with n cars, and let X be the set of all original and potential breaks. For any pair of cars τ_1, τ_2, $1 \leq \tau_1 < \tau_2 \leq n$, we define $X_{(\tau_1,\tau_2)}$ as the set of all original and potential breaks occurring between τ_1 and τ_2, i.e., $X_{(\tau_1,\tau_2)} = X \cap \{(\tau_1, \tau_1+1), (\tau_1+1, \tau_1+2), \ldots (\tau_2 - 1, \tau_2)\}$.

Recovery Model. In many yards, there are certain classification tracks reserved for the sorting procedure considered here, while other tracks are used for different sorting activities. The inital roll-in to the reserved tracks is then scattered over the day, and, when the last train arrives, the other activities are stopped and the first pull-out performed. At this point a scenario is revealed for which the original schedule may be infeasible. With the recovery action of inserting up to k additional sorting steps to the first stage solution, we seek to obtain a feasible schedule for the modified instance. Distributing the recovered solution, i.e. the changed schedule, to all people involved in the operation takes some time depending on the available communication channels. For these reasons, inserting additional sorting steps is only allowed after an offset of p steps.

In terms of classification schedules, which present solutions to our optimization problem, this means the following: given two parameters $p \geq 0$ and $k \geq 0$ and a first-stage solution schedule B of length h, B is to be recovered by inserting up to k additional columns with indices greater than p. This concept is formalized in the following definition.

Definition 1. *Let $B = (b_{h-1}, \ldots, b_0)$ and $B' = (b'_{h-1+j}, \ldots, b'_0)$ be two classification schedules for n cars of length h and $h+j$, $j \geq 0$, respectively. Let further $p \geq 0$ and $k \geq 0$. The schedule B' is called a* (p,k)-extension *of B if $j \leq k$, $b_i = b'_i$ for all $0 \leq i < p$ and $b_{i-j} = b'_i$ for all $p+j-1 \leq i \leq h+j-1$.*

Note that in the definition above the additional columns are all added between the $(p-1)$th and pth step of the original schedule. It can be shown easily that, if inserting k columns at the ith position yields a feasible recovered schedule, inserting these k columns at the $(i-1)$th position instead also yields a feasible schedule. Hence, inserting at the "right-most" allowed position always presents

the most powerful recovery. The notion of (p, k)-extensions yields a natural concept of recoverable robustness as stated in the following definition.

Definition 2. *Let $T_1, \ldots, T_\ell$ be a sequence of ℓ inbound trains and $\mathcal{S}$ a set of scenarios with X^S denoting the corresponding induced set of breaks for every $S \in \mathcal{S}$. A classification schedule B is called* (p, k)-recovery robust *if, for every scenario $S \in \mathcal{S}$, there is a (p, k)-extension of B that is feasible w.r.t. S.*

Most likely, no delay occurs and the inbound trains arrive in the expected order, in which case we usually do not want to apply any recovery for organizational reasons. For our objective this means we look for *feasible* (p, k)-recovery robust classification schedules of minimum length.

In order to specify when a given schedule is (p, k)-recovery robust for a given set of scenarios, we introduce the notion of a *block* of a schedule. A block basically is a maximal set of bitstrings representing integers between two powers of two.

Definition 3. *Let B be a schedule of length h for an inbound train sequence of n cars, and $p \geq 1$. For any bitstring b^j of B, $b^j_{h-1} \ldots b^j_p$ is called the* leading part *of b^j, denoted by $b^j_{>p}$, and $b^j_{p-1} \ldots b^j_0$ the* trailing part *of B, denoted by $b^j_{<p}$. A subset of λ consecutive bitstrings $b^j, \ldots, b^{j+\lambda-1}$ of B is called a* block *of B if their leading parts satisfy $b^{j-1}_{>p} < b^j_{>p}$, $b^j_{>p} = b^{j+x}_{>p}$ for all $1 \leq x \leq \lambda - 1$, and $b^{j+\lambda-1}_{>p} < b^{j+\lambda}_{>p}$, while λ is called the* size *of the block. Furthermore, the jth car of the inbound train sequence is called the* head *of the block.*

The following lemma states the necessary and sufficient conditions for the existence of (p, k)-extensions. The recovery is performed independently for every block, where unresolved breaks are successively fixed by raising the bitstring of the second car of the break and all cars following it up to the end of the block.

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

General Algorithm. Applying the observations of the previous section, we introduce a generic algorithm for computing (p, k)-recovery robust train classification schedules. Basically, the algorithms successively grows the size of a block to its maximum size. The maximum size of a block is determined by two factors: First, a schedule B assigns at most 2^p different bitstrings to the trailing part of cars in the same block, i.e., at most $2^p - 1$ breaks can be resolved. Secondly, the number of unresolved breaks in a block is limited by $2^k - 1$ potential breaks induced by one scenario. We formalize the second condition in the following way.

Definition 4. *Let $T_1, \ldots, T_\ell$ be a sequence of ℓ inbound trains with a total of n cars, $\tau_1, \tau_2 \in \{1, \ldots, n\}$ two cars, and $k \geq 0$. Given a set of scenarios $\mathcal{S}$, a set of breaks $X' \subseteq X_{(\tau_1,\tau_2)}$ is called* k-recoverable *according to $X_{(\tau_1,\tau_2)}$ if $|X' \cap X^S| \leq 2^k - 1$ holds for all $S \in \mathcal{S}$.*

Algorithm 1. k-recovery robust train classification

Data: number of cars n, set of original breaks X_{org}, set of scenarios $\mathcal{S}$, $k, p \geq 0$
Result: k-recovery robust classification schedule B

1 Put $i = 0$, $\tau_i = 1$, $\tau_{\max} = 0$, $X = X' = \emptyset$
2 **while** $\tau_i \leq n$ **do**
3 **while** $\tau_{\max} < \tau_i + 2^p + |X'|$ *and* $\tau_i + 2^p + |X'| \leq n$ **do**
4 Set $\tau_{\max} = \tau_i + 2^p + |X'|$
5 Set $X = X_{(\tau_i, \tau_{\max})} \cap (\cup_{S \in \mathcal{S}} X^S)$
6 Compute a maximum k-recoverable set of breaks $X' \subseteq X$
7 **end**
8 Set $\tau_{\max} = \tau_{i+1} = \min(\tau_i + 2^p + |X'|, n+1)$
9 Compute subschedule of length p for $\tau_i, \ldots, \tau_{i+1} - 1$ feasible w.r.t. $X_{(\tau_i, \tau_{i+1}-1)} \setminus X'$
10 Set $i = i+1$
11 **end**
12 Set $h' = \lceil \log_2 i - 1 \rceil$
13 **for** $j = 0, \ldots, i-1$ **do**
14 Set $b^\tau_{p+h'-1} \ldots b^\tau_p = [j]_2$ for all $\tau_j \leq \tau \leq \tau_{j+1} - 1$
15 **end**
16 **return** B

Algorithm 1 determines the maximum size of a block by repeatedly solving the problem of finding a maximum k-recoverable break set and thus constructs an optimal (p, k)-recovery robust schedule.

Theorem 1. *For any $p \geq 0$ and $k \geq 0$, Alg. 1 computes an optimal (p, k)-recovery robust train classification schedule.*

In Alg. 1 the step of computing a maximum k-recoverable break set in line 6 is not specified. One way of solving this problem is integer programming. As we will show in the following, there is in general no polynomial time algorithm to solve this problem unless $\mathbf{P} = \mathbf{NP}$.

Computational Complexity. In this section we assume w.l.o.g. that we are looking for a maximum k-recoverable break set for the cars $1, \ldots, n$, i.e., let $\mathcal{S}$ be a set of scenarios, find a maximum k-recoverable break set X' of $X = \cup_{S \in \mathcal{S}} X^S$. By a reduction from the independent set problem, the decision version of this problem is strongly **NP**-hard for $k = 1$. A different reduction from 2^kSAT leads to the **NP**-completeness for any constant $k \geq 2$.

Theorem 2. *Let $T_1, \ldots, T_\ell$ be a sequence of ℓ inbound trains, $\mathcal{S}$ a set of scenarios, and $K \geq 0$. For any constant $k \geq 1$, it is strongly* **NP***-complete to decide whether there exists a k-recoverable break set of size K.*

This theorem not only states that Alg. 1 will only run in polynomial time if $\mathbf{P} = \mathbf{NP}$ but also enable us to prove the **NP**-completeness of the (p, k)-recovery robust classification problem.

Corollary 1. *Let $T_1, \ldots, T_\ell$ be a sequence of ℓ inbound trains, $\mathcal{S}$ a set of scenarios, $h, p \geq 0$, and $k \geq 1$ const. Deciding whether there is a feasible (p,k)-recovery robust classification schedule of length at most h is an* **NP***-complete problem.*

Infeasible Initial Solutions. In our model the first-stage solution is a feasible classification schedule for the original order of trains. A special case of this setting is to allow recovery even in case of no disturbance. In this case the original breaks can be modeled by a scenario S_{org} with $X^{S_{\text{org}}} = X_{\text{org}}$ and no original breaks are there, i.e., we assume that the cars arrive in perfect order.

4 Limited Number of Delayed Trains

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

Scenario Model. Given some some parameter j, up to j trains are delayed each by an arbitrary amount: let $\Theta = T_1, \ldots, T_\ell$ be an inbound train sequence and $\Theta^\sigma = T_{\sigma^{-1}(1)}, \ldots, T_{\sigma^{-1}(\ell)}$ be an order of trains induced by some permutation $\sigma\colon [\ell] \to [\ell]$. Then, a sequence $\bar{\Theta} = T_{\bar{\sigma}^{-1}(1)}, \ldots, T_{\bar{\sigma}^{-1}(\ell)}$, where $\bar{\sigma}$ is some permutation, is called an (α, k)-*delayed sequence* of Θ^σ if $\sigma(\alpha) < k$ and the following conditions hold: $\bar{\sigma}(x) = \sigma(x)$ if $\sigma(x) < \sigma(\alpha)$ or $\sigma(x) > k$, $\bar{\sigma}(x) = \sigma(x) - 1$ for $\sigma(\alpha) < \sigma(x) < k$, and $\bar{\sigma}(\alpha) = k$. Less formally, train T_α is delayed from the $\sigma(\alpha)$th to the kth position. The set of scenarios $\mathcal{S}_j$, $0 \leq j \leq \ell$, is now defined to contain a scenario S (inducing some sequence Θ^S) iff there is a sequence $\Theta^0, \ldots, \Theta^j$ of train sequences Θ^i such that $\Theta^0 = \Theta$, Θ^i is an (α_i, k_i)-delayed sequence of Θ^{i-1} for all $i = 1, \ldots, j$, and $\Theta^j = \Theta^S$. Every train T_{α_i} will furthermore be called to be *delayed* by S.

Dominating Set of Scenarios. We will see in Thrm. 3 that our considerations can be restricted to the dominating subset $\bar{S}_j \subseteq \mathcal{S}_j$ of scenarios defined as follows: a scenario S is a member of $\bar{S}_j$ iff there is a sequence $\Theta^0, \ldots, \Theta^j$ of train sequences Θ^i such that $\Theta^0 = \Theta$, Θ^i is an (α_i, ℓ)-delayed sequence of Θ^{i-1} for all $i = 1, \ldots, j$, $\alpha_i < \alpha_{i-1}$ for all $i = 1, \ldots, j$, and $\Theta^j = \Theta^S$. In other words, if two trains are delayed by $S \in \bar{S}_j$, they swap their relative order and arrive later than all punctual trains. Note that for uniquely defining a scenario $S \in \bar{S}_j$ it suffices to list the j delayed trains since the order and amount of their delay is determined by the definition of $\bar{S}_j$.

Theorem 3. *Given any $p, k, j \geq 0$, let B be a feasible (p,k)-recovery robust schedule for $\bar{S}_j$. Then, B is a feasible (p,k)-recovery robust schedule for $\mathcal{S}_j$.*

Any potential break $(\tau, \tau{+}1)$ can only be induced by S if the train containing τ is delayed, but also the converse implication holds for $\bar{S}_j$ as stated in the following lemma.

Lemma 2. *Let $T_1, \ldots, T_\ell$ be a sequence of inbound trains and $S \in \bar{S}_j$ some scenario. For any potential break $\beta = (\tau, \tau+1)$ with $\tau \in T_x$, $x \in \{1, \ldots, \ell\}$, $\beta \in X^S$ iff T_x is delayed by S.*

Algorithm 2. Max. k-Recoverable Set of Breaks for $\mathcal{S}_j$ with Unique Cars

Input: Parameters $j, k \in \mathbb{N}$ and sets of induced breaks $X_1, \ldots, X_\ell$
Output: Maximum recoverable set of breaks

```
Descendingly sort X_1, ..., X_ℓ such that |X_{i_1}| ≥ |X_{i_2}| ≥ ... ≥ |X_{i_ℓ}|
Put α := max{i_t : |X_{i_t}| = |X_{i_1}|}
while Σ_{t=1}^{j} |X_{i_t}| ≥ 2^k do
    Remove an arbitrary break from X_{i_α}
    Put α := max{i_t : |X_{i_t}| = |X_{i_1}|}
end
return ⋃_{i=1}^{ℓ} X_i
```

As an immediate consequence, the set of potential breaks X^S of any scenario $S \in \bar{S}_j$ can be partitioned into disjoint subsets w.r.t. the respective delayed train causing the break, a fact which is applied in the algorithm of the following section. We will call the set $X_i := \{(\tau, \tau+1) | \tau \in T_i, \exists y > i : \tau+1 \in T_y\}$ the *set of breaks induced by train* T_i.

Maximum Recoverable Sets of Breaks. For $\bar{S}_j$ a maximum recoverable set of breaks is computed with Alg. 2: we repeatedly resolve potential breaks of the train that induces the highest number of unresolved breaks until the worst case scenario does not exceed the recovery capability given through the parameter k. Correctness, optimality, and the running time of Alg. 2 are summarized in the following theorem.

Theorem 4. *Given a set of potential breaks X for some classification instance with inbound trains $T_1, \ldots, T_\ell$, a maximum k-recoverable set of breaks $X' \subseteq X$ w.r.t. $\bar{S}_j$ can be computed in polynomial time.*

As an immediate consequence of Thrm. 4, the problem of train classification can be solved in polynomial time by combining Alg. 2 into Alg. 1. The resulting algorithm is implemented in the following section and tested for a number of real-world classification instances.

Experimental Evaluation. For the evaluation of the algorithm just described, we took the five real-world instances used in [10], which unfortunately are the only real-world instances available to us. They correspond to five days of traffic in the Swiss classification yard Lausanne Triage, with volumes ranging from 310 to 486, numbers of inbound trains between 44 and 49, outbound trains between 24 and 27, and numbers of breaks between 24 and 28. In order to obtain unique types of cars, we converted all cars of the same type between two consecutive original breaks to distinct types ascending in the order the cars appear between the breaks. The algorithm was implemented in C++, compiled with GNU g++-4.4, and run on an 1.8 GHz Intel Core Duo CPU with 2 GB main memory.

Essentially, through adjusting the parameters p, k, and j, the algorithm allows flexibly trading off shortest schedules against the other extreme of strict robustness. Given some train classification instance, let $\underline{h}$ denote the length of

an optimal non-robust schedule and $\bar{h}$ the length of an optimal strictly robust schedule. The values $\underline{h}$ and $\bar{h}$ present the lower and upper bounds for the length resulting from any combination of j, k, and p. Yet, as explained in Sect. 1 (Related Work), $\bar{h}$ may be exceeded by the geometric method, i.e. an optimal strictly robust schedule disregarding presorted inbound trains, and even longer schedules than this are obtained by triangular sorting.

Table 1. Optimal length values for S_j with (p, k)-extensions for the five traffic instances: the values for the triangular and geometric method are given in the first and second column, resp., $\bar{h}$ and $\underline{h}$ in the third and fourth column, resp. Omitted entries represent no meaningful choice of p.

k	t	g	0	0	1								2																3																4		
p	r	e	0	0	0		1		2		3		4		0		1				2						3																1		2	
j	i	o	1	0	1	2	1	2	1	2	1	2	1	2	3	4	1	2	3	4	1	2	3	4	2	3	4	3	4	7	8	5	6	7	8	3	4	5	6	7	8	4	5	6	7	8	≥ 1	13	≥ 14
inst-1	11	6	5	2	2	4	3	4	3	5	4	5	4	5	2	3	2	3	3	4	2	3	3	4	3	4	5	4	5	2	3	2	3	3	3	2	3	3	3	3	4	3	3	3	4	4	2	2	2
inst-2	8	5	5	3	3	4	3	4	3	4	3	5		5	3	3	3	3	3	3	3	3	3	4	3	4	4	4	4	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	4	3	3	3
inst-3	8	5	5	2	2	4	2	4	3	4	3	4		5	2	3	2	2	2	3	2	3	3	4	3	3	4		4	2	3	2	2	2	3	2	2	2	3	3	3		3	3	4	2	2	2
inst-4	10	6	5	2	2	4	2	4	3	4	3	5		5	2	3	2	2	2	3	2	3	3	4	3	3	4		5	2	2	2	2	2	3	2	2	2	3	3	3		3	3	4	2	2	3
inst-5	9	6	5	2	2	4	3	4	3	4	3	5		5	2	3	2	3	3	3	2	3	3	4	3	3	4		4	2	2	2	2	2	3	2	2	2	3	3	3		3	3	4	2	2	2

Table 1 summarizes the computed length of an optimal recovery robust schedule according to the different parameters p, k and j. As lower and upper bounds for those length inst-2 requires $\underline{h}= 3$, while all other instances yield $\underline{h}= 2$ and $\bar{h} = 5$ for all instances. The geometric method requires $h = 6$ for three of the instances, and the triangular method even between eight (int-2 and inst-3) and eleven steps (inst-1), which shows that ignoring presorted input wastes a lot of potential for improvement.

If only small amounts of recovery action ($k = 1$) are allowed, for $j = 1$ the schedule length does not exceed $\underline{h}$ for inst-1 and inst-5 with $p = 0$, for inst-3 and inst-4 with $p \leq 1$, and for inst-2 even for $p \leq 3$, so yet for lowest degrees of recovery we obtain some robustness without increasing the length beyond that of an optimal non-robust schedule. Raising the degree of disturbance to $j \geq 2$, we still obtain a length $h = 4 < \bar{h}$ if the value of p is increased to $p = 1$ for inst-1, to $p = 2$ for inst-2, inst-4, and inst-5, and even to $p = 3$ for inst-3. These values are significantly smaller than those for the strictly robust methods of geometric or even triangular sorting.

The degree of robustness grows rapidly with increasing degrees of recovery, and for $k = 4$ with $p \leq \underline{h}$—except for inst-4 with $p = 2$—we can allow any number of delayed trains and still achieve the length $\underline{h}$ of an optimal robust schedule. Between these extremes, Tab. 1 shows how far the value of p can be raised for $k = 2$ and arbitrarily high amounts of delay $j \geq 4$: most instances allow $p = 1$ to obtain $h = 3$, and $h = 4$ can be achieved even for $p = 4$ for three out of five instances. For $k = 3$, Fig. 2 (left) summarizes the values of inst-1: a schedule of length 3 with a recovery action starting after the third sorting step suffices to cope with a delay of up to six trains and $p = 1$ allows $h \leq 3$ even for any disturbance value j. Similarly, for a fixed value of $p = 2$, Fig. 2 (center) shows the rapid growth of robustness: except for inst-2, k must be raised rather

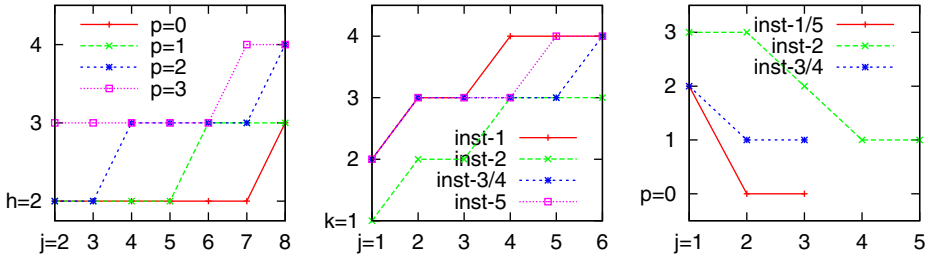

Fig. 2. left: optimal schedule lengths h of inst-1 for $k = 3$; center: highest possible values of k to achieve a length of $\underline{h}$ for $p = 2$; right: smallest possible values of p to achieve a length of $\underline{h}$ for $k = 2$;

quickly between $j = 1$ and $j = 3$ to achieve a length of $\underline{h}$, whereas the required value of k does not exceed four for higher disturbances $j \geq 6$. Conversely, Fig. 2 (right) fixes $k = 2$ and shows the maximum value of p that allows a length of $\underline{h}$: $j = 1$ still allows $p = \underline{h}$ for all instances, but, except for inst-2, this length $\underline{h}$ cannot be achieved for any choice of p for high amounts of delay $j \geq 4$. Hence, higher values of k contribute much more to the potential of recovery than low values of p. Summarizing, through adjusting the recovery parameters k and p, our algorithm presents a tool to flexibly trade off between fast classification and robust schedules and, even for high degrees of robustness, we achieve much shorter schedules than the triangular method currently applied in practice.

5 Conclusion

We have developed a practically applicable algorithm for deriving robust train classification schedules of minimum length. In contrast to [2], we regard multiple inbound and outbound trains, which allows integrating the most relevant disturbance in form of delayed trains. We have introduced the natural recovery action of (p, k)-extensions, for which we proved that the problem is **NP**-complete for every constant $k \geq 1$. Nevertheless, for the simple yet quite genral set of scenarios S_j, we have shown our generic algorithm of Sect. 3 can be implemented in polynomial time by solving the subproblem of calculating a maximum recoverable set of breaks efficiently. The experimental study of Sect. 4 indicates that the resulting algorithm improves on the current classification practice as it yields shorter schedules and still allows high degrees of robustness. Its flexibility further allows balancing between strictly robust and optimal non-robust schedules and raises potential for increased traffic throughput in classification yards.

Future Work. Further practical restrictions, such as a limited number of classification tracks (see [15]), are desirable to be considered in the context of robustness. In a practical settings where the actual sorting is started (through the first pull-out) before all inbound trains have arrived, the online version of the problem becomes relevant. Moreover, the number of cars rolled in presents a

secondary objective, which can be additionally minimized for a minimum length. Finally, making the order of inbound trains part of the optimization yields different robust optimization problems.

References

1. Cacchiani, V., Caprara, A., Galli, L., Kroon, L., Mároti, G.: Recoverable robustness for railway rolling stock planning. In: ATMOS 2008, IBFI, Schloss Dagstuhl (2008)
2. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A.: Robust algorithms and price of robustness in shunting problems. In: Liebchen, C., Ahuja, R.K., Mesa, J.A. (eds.) ATMOS 2007. IBFI, pp. 175–190. Schloss Dagstuhl (2007)
3. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A., Schachtebeck, M., Schöbel, A.: Recoverable robustness in shunting and timetabling. In: Ahuja, R., Möhring, R., Zaroliagis, C. (eds.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 28–60. Springer, Heidelberg (2009)
4. Cicerone, S., Di Stefano, G., Schachtebeck, M., Schöbel, A.: Dynamic algorithms for recoverable robustness problems. In: ATMOS 2008. IBFI, Schloss Dagstuhl (2008)
5. Daganzo, C.F., Dowling, R.G., Hall, R.W.: Railroad classification yard throughput: The case of multistage triangular sorting. Transp. Res. 17A(2), 95–106 (1983)
6. Dahlhaus, E., Manne, F., Miller, M., Ryan, J.: Algorithms for combinatorial problems related to train marshalling. In: AWOCA 2000, pp. 7–16 (2000)
7. Flandorffer, H.: Vereinfachte Güterzugbildung. ETR RT 13, 114–118 (1953)
8. Gatto, M., Maue, J., Mihalak, M., Widmayer, P.: Shunting for dummies: An introductory algorithmic survey. In: Ahuja, R., Möhring, R., Zaroliagis, C. (eds.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 310–337. Springer, Heidelberg (2009)
9. Hansmann, R.S., Zimmermann, U.T.: Optimal sorting of rolling stock at hump yards. In: Mathematics - Key Technology for the Future: Joint Projects Between Universities and Industry, pp. 189–203. Springer, Heidelberg (2007)
10. Hauser, A., Maue, J.: Experimental evaluation of approximation and heuristic algorithms for sorting railway cars. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 154–165. Springer, Heidelberg (2010)
11. Jacob, R., Márton, P., Maue, J., Nunkesser, M.: Multistage methods for freight train classification. In: Liebchen, C., Ahuja, R.K., Mesa, J.A. (eds.) ATMOS 2007, pp. 158–174. IBFI, Schloss Dagstuhl (2007)
12. Jacob, R., Márton, P., Maue, J., Nunkesser, M.: Multistage methods for freight train classification. Networks (2010) (to appear, 2010)
13. Krell, K.: Grundgedanken des Simultanverfahrens. ETR RT 22, 15–23 (1962)
14. Liebchen, C., Lübbecke, M.E., Möhring, R.H., Stiller, S.: The concept of recoverable robustness, linear programming recovery, and railway applications. In: Ahuja, R., Möhring, R., Zaroliagis, C. (eds.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 1–27. Springer, Heidelberg (2009)
15. Márton, P., Maue, J., Nunkesser, M.: An improved classification procedure for the hump yard Lausanne Triage. In: Clausen, J., Di Stefano, G. (eds.) ATMOS 2009. IBFI, Schloss Dagstuhl (2009)
16. Pentinga, K.: Teaching simultaneous marshalling. Railway Gaz, 590–593 (1959)
17. Siddiqee, M.W.: Investigation of sorting and train formation schemes for a railroad hump yard. In: Proc. of the 5th Int. Symposium on the Theory of Traffic Flow and Transportation, pp. 377–387 (1972)