On shunting over a hump

Riko Jacob

Institute of Theoretical Computer Science, ETH Zurich, Switzerland
ETH Zentrum, CH-8092 Zürich
rjacob@inf.ethz.ch

Abstract. We consider the situation of rearranging freight trains in a shunting yard with a hump and several tracks. Here, an elementary shunting operation is to push a all cars of one track over the hump. On the way back to the tracks the cars can be guided individually to an arbitrary track. The goal is to create one or several trains that have their cars in a particular order. We measure the efficiency of a shunting plan by the number of elementary shunting operations, and to a lesser extent by the total number of car movements.
We find optimal shunting operations in the following situations:

- The incoming trains are unsorted and the shunting yard has sufficiently many tracks but the length of the tracks is limited (capacities).
- The order of the cars in a single incoming train is given, the tracks of the shunting yard are sufficiently long, but there is only a limited number of tracks.

In contrast, we show that the following problems are $\mathcal{NP}$-complete:

- There are several incoming trains and for each the order of the cars is given, but the order in which to start shunting the trains can be chosen.
- There is one train with given order of the cars, and the lengths of the tracks is limited.

These results are based on expressing the shunting task as finding a suitable set of codes.
The results will perhaps not solve the practical problem directly, but rather provide some first structural insights and complexity limits that can guide the search for practical algorithms.

1 Introduction

One central task in a freight railway operation is rearranging trains in a shunting yard, also called switching in a classification yard. This paper is concerned with the theoretical foundations of the shunting task for a certain type of shunting yard, that is characterized by the availability of a hump over which a pseudotrain of cars can be pushed and distributed to different tracks. Here, we make the situation mathematically precise, explore special cases for which an optimal solution can be given, and identify $\mathcal{NP}$-hard questions. Since the task at hand is sorting on an unusual model of computation and with specific restrictions, it is not too

surprising that many of the basic concepts of theoretical computer science turn out to be relevant, like sorting, binary codes and Fibonacci numbers. The results presented here are not meant to directly solve any of the planning problems occuring in practice, but rather provide an understanding of the mathematical structure of the problem. At this stage, we find several structural insights into this specific type of shunting operation, in particular situations where we can easily describe the optimal operations. This does not solve the operations research question how to compute a good shunting plan, but it is certainly an important starting point for practical algorithms that work for the more complicated situations one expects to find in practice.

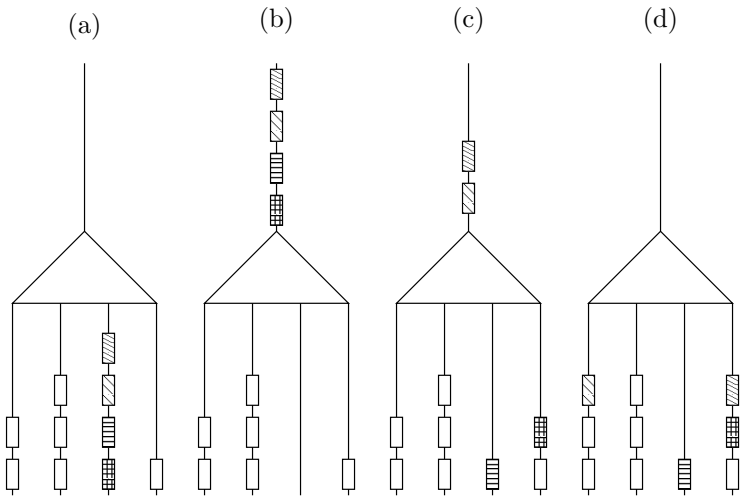

Fig. 1. The different stages of a track pull operation; stage (c) is after two cars have been pushed over the hump, stage (d) is the final configuration.

Setting Here, we introduce the basic concepts of the situation, including the typical variable name for certain quantities.

The presented results assume the following classical layout of a shunting yard: There is a certain number w of parallel classification tracks, also called classification bowl, that can be accessed only from the same side, i.e., these tracks are dead ends. The access to the tracks is over a single hump, and there is a collection of switches, called the ladder, between the hump and the classification tracks. This layout supports a sorting operation by repeatedly doing the following, which we call a *track pull* (operation), as visualized in Figure 1:

- Connect the cars of one classification track into a pseudotrain (a)
- Pull the pseudotrain over the hump (b)
- Disconnect the cars in the pseudotrain

– Push the pseudotrain slowly over the hump, yielding single cars that run down the hill from the hump towards the classification tracks
– Control the switches such that every single car goes to a specified track (c,d)

Our main complexity measure for a shunting operation is the number of track pulls t that are performed to achieve a certain (set of) train(s). We assume that the order of the cars in such an *outgoing train* is completely specified as the goal of the operation. The total number of times a car is pushed over the hump, that is, the sum of the lengths of all pseudotrains, is usually denoted by m, the number of *car pulls*.

A concrete shunting yard can be described by its *width* w, the number of classification tracks, and the lengths of these classification tracks. We denote by c the number of cars that fit on all tracks. We usually assume that all tracks have the same length, and that all cars occupy the same amount of space on the track.

The number of cars in a train is usually denoted by n.

Related Work The starting point of the considerations here was a collaboration with the freight section of Swiss Federal Railways [5]. There, we show $\mathcal{NP}$-completeness for a sequencing problem of the incoming and outgoing trains with the aim of limiting the number of cars that are simultaneously in the shunting yard. This problem does not address any specific operations at the yard.

It seems that the operations research and computer science literature barely discusses the reclassification operation at a shunting yard with a hump. A shunting yard with the same layout is discussed in [2]. However, there one step of the shunting operation consists of concatenating *all* classification tracks into one pseudotrain, which is then pushed over the hump, called a *humping step*. For a shunting yard with w classification tracks, this operation can be easily simulated by w track pulls (our measure): The first $(w-1)$ tracks are pulled in the correct order and all cars are guided onto track w, which is finally pulled and the humping step is performed. Given the difference in the operation and cost model it is not surprising that the radix-sort based scheme of [2] achieving $\log_w n$ humping steps translates into a shunting scheme in our model with cost $w \log_w n$, which is suboptimal by a factor $w/\log_2 w$ in our cost model. Situations where the order of the outgoing train is not completely specified are discussed in [1, 2], where the respective optimization problem is shown to be $\mathcal{NP}$-complete.

Other work that considers shunting operations does not address the specific situation of a shunting yard with a hump, like [3, 6].

Results and structure of the paper First, in Section 2, we model the shunting task as a problem of finding a set of binary codes. This immediately yields a optimal shunting plans if the shunting yard is sufficiently big. Then, in Section 3, we consider the situation that the shunting yard has sufficiently many different track, but there is a uniform capacity limit for these tracks, and the incoming train is not ordered (or reversed). Again, it is possible to produce an optimal shunting plan. Also for the situation where the number of tracks is limited, but

the capacity of each track is sufficiently big, it is possible to produce an optimal shunting plan, as detailed in Section 4.

One situation that leads to an $\mathcal{NP}$-hard optimization problem is discussed in Section 5, namely the situation of a presorted train and a shunting yard with limited track length. Here, we also find that it is possible to approximate the necessary number of track pulls within a factor of 2 in polynomial time.

Not surprisingly, all of the optimization problems considered in this paper can easily be seen to be in $\mathcal{NP}$. This amounts to the observation that the capacity restrictions and correctness restrictions translate into easy conditions on the codes that represent a shunting plan.

2 Preliminaries

First, we should describe a precise formulation of a plan for a shunting task on a shunting yard with a hump. In general, such a plan has to specify a sequence of track pull operations, given by the track whose cars are pulled, and for every car which track it is sent to. We will name the tracks according to the time they are pulled, i.e., $T = \{1, \ldots, h\}$. This means that one physical track might get several such names (numbers) if it is pulled several times during the shunting plan. In such situations, the logical track is annotated by the name of a physical one. Of course, if there is no limit on the number of tracks ($w \geq h$), there is no need to reuse a track, and this annotation by names of physical tracks is not necessary. With this numbering of the tracks, the itinerary of a car can be described by the sequence of logical tracks it visits. For the task at hand, it is convenient to specify this sequence as a bitstring or code $b_1 \cdots b_h$ where the different bits stand for the logical tracks, and there is a 1 if and only if the car visits that track. Now, if track i is pulled, the new destination of a car is given by its next 1 in its code, i.e., the lowest index $i' > i$ with $b_{i'} = 1$.

In a situation where the order of the cars in one train is rearranged, naturally the incoming train is pulled first, and the track on which the outgoing train is composed is pulled last. Because all cars visit these two logical tracks, we can omit them from the code, such that the number h stands for the intermediate tracks.

Now let us analyze the (outgoing) train resulting from the shunting operations specified by a set of codes. Consider two cars that have assigned the codes a and b. If the two cars have the same code $a = b$, they will have the same order in the outgoing train as in the incoming train. Now assume that the cars of the outgoing train on the logical classification h are numbered from car 1, the one furthest from the hump to n, the one closest to the hump. Then, two cars that are consecutive in the outgoing train can get the same code if they are in the correct order in the incoming train. A maximal interval of cars in the outgoing train that have this property are called a *run* (similar to a chain of [2]), and the split of the cars into runs can easily be computed by scanning over the outgoing train and starting a new run whenever two cars are in the wrong order in the incoming train.

If the two codes a and b are different, the relative position of the two cars in the outgoing train is given by the highest (numbered) bit where the two codes are different, and the car that has the 1 is closer to the hump. This is in fact the same as considering the codes as binary representations of numbers by $C = \sum_{i=1}^{h} c_i 2^{i-1}$. With this, the outgoing train is given by the values of the codes, the car furthest from the hump has the smallest value, the one closest to the hump the highest.

These definitions immediately lead to the following characterization of the number of track-pulls that are necessary to reorder a given train, if the shunting yard is sufficiently big.

Theorem 1. *An incoming train that consists of r runs as defined by the outgoing train can be shunted over a hump in $h = \lceil \log_2 r \rceil$ intermediate track pulls, assuming there are h physical tracks, each of which long enough to hold the complete train.*

Proof. No two cars of different runs can have the same code. By assigning the codes corresponding to increasing binary numbers to the runs, a correct shunting plan is specified. There are 2^h different binary codes with h bits.

An alternative proof of the lower bound in Theorem 1 extends the notion of runs to an arbitrary intermediate situation in the shunting yard. When considering the cars of the outgoing train in their natural order (towards the hump) we define a run to stop for every backward jump on a track, and for every jump to a different track. Now a single track pull operation can unite many runs, but each resulting run stems from at most two runs before the track pull: If there would be three different runs that are united into one two of these runs would have to reside on the same track, which is impossible for different runs. Now the best case is that all the runs on the currently pulled track unite with one other run, and the number of runs halves with every track pull. This also yields a lower bound $h = \lceil \log_2 n \rceil$.

3 Restricted Capacity

In this section we consider the case where all intermediate tracks can hold up to c cars. There is one physical track per logical track, i.e., we assume that the shunting yard consists of sufficiently many classification tracks.

3.1 Unordered Incoming Train

Assume that the order of the cars in the incoming train is not known in advance, or equivalently, that the order of the cars in the incoming train is the reverse of the order in the outgoing train, and that hence all runs consist of a single car.

To be precise, we assume that there is one track that is long enough to hold the complete train, and this track is only used to assemble the outgoing train,

but not to be pulled over the hump. Similarly, the incoming train resides on a long track that cannot be used in the later shunting operations.

This naturally leads to the following question: What is the maximal length of an (unordered) train that can be sorted (reversed) with h track pulls, where each track holds at most c cars? In the world of the codes, this translates into the question: what is the maximal size of a set of binary codes over h positions, such that the total number of ones at any position is at most c.

First, we discuss an algorithm that produces one such optimal set of codes. Then we also discuss the asymptotics.

Algorithm to produce an optimal code Let us first make a detour to a slightly different objective, namely the total number of car pulls m. In the setting of the tracks of limited capacity c, it is clear that pulling h tracks can pull at most a total of $m = h \cdot c$ cars. In the world of the codes, this means that the total number of ones used in the set of codes is limited by $h \cdot c$. By a knapsack argument, it is optimal to only use codes with a certain number ones if all codes with fewer ones are used. Hence, if we find a subset of codes that usese all codes with less than ℓ ones, some codes with precisely ℓ codes and no codes with more than ℓ ones, for some number ℓ, and the total number of ones is between $m - j + 1$ and m, then this subset achieves the maximal number of different codes subject to the constraints h and c.

Let $K(h, m)$ stand for the maximal number of different bitcodes of length h that have a total number of ones less than m.

Theorem 2. *For integers c, h, if $c \leq 2^h$ then there is a set of $K(h, h \cdot c)$ bitcodes of length h such that at every position at most c codes have a one.*

The following Lemma implies Theorem 2, and also gives an efficient way to compute such a set of codes.

Lemma 1. *For positive integers h, n, i, $h \geq i$. Assume $n \leq \cdot \binom{h}{i}$. Then there is a set of n codes of length h with precisely i ones in each, such that the total number of ones at two positions differs by at most one. That is, there are number $h_1 + h_2 = h$ and $c = \lceil \frac{ni}{h} \rceil$ such that in h_1 positions the total number of ones is c, and in h_2 positions the total number of ones is $c - 1$.*

Proof. By induction on h, using well known identities of binomial coefficients.

In one inductive step a set S_1 of c codes with ones in the last position is created. The remaining ones of the codes in S_1 are equally distributed over all remaining positions. Again inductively, the remaining space is filled with codes with i ones.

If $h = i$, the assumption about n implies $n = 1$. The code consisting of $i = h$ ones (and no zeros) shows the statement of the lemma.

Now assume $h > i$ and inductively assume that the lemma holds for all $h' < h, h' \geq i$. We construct a set of codes as the (disjoint) union of two sets S_1 and S_2, where $|S_1| = c$, all codes in S_1 have a one at the last position, whereas all codes

in S_2 have a zero in the last position. This leads to two (recursive) applications of the lemma, namely to chose the remaining $i-1$ ones for the codes of S_1, and the whole codes in S_2. To that end, we have to decide upon where the remaining ones of S_1 and S_2 should be placed. Fitting to the formulation of the lemma, we choose to distribute the ones of S_1 equally among all the remaining $h-1$ positions, such that also the ones of S_2 can be equally distributed. It remains to show that the required numbers of ones, numbers of positions, and numbers of codes fulfill the assumptions of the lemma.

To this end, we define some abbreviations with the following naming conventions: Primed values are associated with the invocation of the lemma for S_1, double primed values for S_2. 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. Such a value is marked by a hat.

Note that the bound on n implies $c \leq \binom{h-1}{i-1}$ (an alternative way to describe the assumption of the lemma) because of $ni \leq i \cdot \binom{h}{i} = \frac{h}{i} \cdot i \cdot \binom{h-1}{i-1}$ which reads as $c \leq \binom{h-1}{i-1}$). Let us define $e = \binom{h-1}{i-1} - c$. Because we are interested in the remaining positions, we define the average goal height for these positions only, and not for all positions: $\hat{c} = \frac{h_1-1}{h}c + \frac{h_2}{h}(c-1) = \frac{ni-c}{h-1}$, and the corresponding $\hat{e} = \binom{h-1}{i-1} - \hat{c}$. Observe that $e \leq \hat{e} (\leq e+1)$.

The average height resulting from S_1 on the remaining positions is necessarily $\hat{c}' = c(i-1)/(h-1)$. For S_2 we hence get $\hat{c}'' = \hat{c} - \hat{c}'$.

Now we invoke the lemma twice inductively (recursively). Once with $h' = h-1$ code length, $i' = i-1$ number of ones per code, and $n' = c$ as the number of codes, resulting in S', where the positions with more ones are higher numbered. We invoke it another time with $h'' = h-1$ as code length, $i'' = i$ ones per code, and $n'' = n-c$ codes, resulting in the set of codes S'', where the positions with more ones are lower numbered. Assuming, as we will argue for later, that the assumptions of the lemma are fulfilled, we can now create the required set of codes. We extend S' to S_1 by appending a one, and S'' to S_2 by appending a zero. By the sorting of the positions in S' and S'', there are only two types of positions, and their respective total number of ones differs by at most one. Additionally, the higher of these numbers is c if they are different, and otherwise $c-1$. Hence the statement of the lemma is achieved.

Now, lets verify that the assumptions of the inductive invocations of the lemma are satisfied. With the alternative formulation of the assumption, we should check $\hat{c}' \leq \binom{h-2}{i-2}$ for the first invocation: By definition $\hat{c}' = c(i-1)/(h-1) = \frac{i-1}{h-1}\binom{h-1}{i-1} - \frac{i-1}{h-1}e = \binom{h-2}{i-2} - \frac{i-1}{h-1}e$.

For the second invocation we should check $\hat{c}'' \leq \binom{h-2}{i-1}$. So, $\hat{c}'' = \hat{c} - \hat{c}' = \binom{h-1}{i-1} - \hat{e} - \binom{h-2}{i-2} + \frac{i-1}{h-1}e = \binom{h-2}{i-1} + \frac{i-1}{h-1}e - \hat{e} \leq \binom{h-2}{i-1}$, where the last inequality relies upon $h \geq i$ making the coefficient of e smaller than one, so that this whole term is at most $\hat{e}$ (by $e \leq \hat{e}$).

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

3.2 Connection to Entropy

We have seen the connection between a set of codes and the shunting plan. This means in particular, that it is sufficient to identify a particular car (or run) to know which tracks (in time) this car visits. Now, we can consider the random experiment of uniformly choosing a car, leading to the h binary indicator variables X_i, where a one means that the chosen car visits track i. Because a single variable has an entropy of at most 1, whereas the total entropy of the experiment (choosing a car) has entropy $\log n$, we have an alternative proof that it is impossible to achieve the sorting in less than $\log n$ steps. Again, we see that this bound is tight for n being a power of two, such that all variables have entropy precisely one (there are precisely $n/2$ cars that visit the track), and all variable are independent (even all pairs of disjoint sets of variables).

Now, if there is a capacity constraint k on all tracks, this means that the entropy of a single variable is at most the entropy of a k/n biased coin, i.e., $E(\frac{k}{n}) = \frac{k}{n}\log\frac{n}{k} + \frac{n-k}{n}\log\frac{n}{n-k}$. This leads to the lower bound on h, the number of used tracks, of $\frac{\log n}{E(k/n)}$. Naturally, the lower bounds on the number of tracks used is weaker than what comes out of the optimal code of the previous section, but in return it is a nice analytical bound. In many situation it is actually quite good, the next paragraph discusses an example.

Consider the situation where the optimal code uses half the codes with two ones. More precisely, we consider any h such that $h-1$ is a multiple of four, define $n = h + \frac{h(h-1)}{4}$, and set the capacity constraint $k = \frac{h\cdot 1 + \frac{h(h-1)}{4}\cdot 2}{h} = 1 + \frac{h-1}{2}$. By Theorem 2, this is an optimal set of codes. Now, we compare the lower bound resulting from the entropy with the (optimal) value of h. Asymptotically, for large values of h, the ration between this lower bound and h is 1, and for $h \geq 2$ this ratio is bigger than .5.

4 Restricted Width

In this section, we consider the case where one train should be sorted, every track can hold the complete train, but the number of tracks in the shunting yard is limited by $w \geq 2$. We assume that the incoming train is located on one of the tracks, and that the outgoing train can be created at any of the tracks. To simplify the exposition, we slightly deviate from the notation in the other sections and assume that pulling the incoming track is counted as a track-pull (all codes start with a 1), and that the track of the outgoing train is also counted and part of the code (all codes end with a 1). The number of track pulls is h, and the case $h-1 \leq w$ is basically that of Theorem 1 for $h-2$. For the sake of concreteness, assume the tracks are called $T_1, \ldots, T_w$, and that the incoming train is located on track T_1.

We can represent a shunting plan in this situation by binary codes of length h and the order in which the tracks are pulled, i.e., a sequence of track names of length h. By defintion we have that the first pulled track is the input track T_1, and the last pulled track is the output track, such that all codes have a one at

the first and at the last position. Now, the binary codes have length h, but are restricted by the destination track being available. More precisely, assume that a code has a one at a certain position. Then there are precisely w next choices for tracks, namely the first occurence of a t_i in the remaining sequence of track pulls, and these are the only possibilities for the next one. For example, if the tracks are pulled in a round robin fashion there may not be w consecutive zeros in any of the codes.

Let us analyse the number of runs that can be shunted with h track pulls on w tracks that are used round-robin, called R_h. We have $R_1 = R_2 = 1$, and $R_h = 2^{h-2}$ for $3 \leq h \leq w$ because there are $h-2$ positions with an unrestricted binary code.

Then, we get the recursion $R_h = \sum_{i=h-w}^{h-1} R_i$ for $h > w$: All valid codes with length h start with a one, then have the next 1 at a position in the range $h-w$ to $h-1$. Now the number of such codes is the sum of the number of codes starting with a 1 at this particular position, having a trailing 1, and no w consecutive zeros.

For $w = 2$ these numbers are the Fibonacci numbers F_i, for larger values of w a generalization of them. In any case we have $F_h \leq R_h \leq 2^{h-2}$.

Now it remains to be shown that it is optimal to pull the tracks in a round robin fashion. We will do this inductively. Of course for $h \leq w$ this is the case. Assume we already know that the maximal number of codes on h' positions (for the best possible track sequence) is $R_{h'}$ for all $h' < h$. Now take one optimal track sequence and set of codes for h track pulls. The codes divide into at most w classes by their second one (the positions depend on the track sequence). Order the classes according to this position of the second one. Then, the first class has codes of length at most $h-1$, the second of length at most $h-2$, and so on. Hence, the number of codes in the classes is bounded by R_{h-1}, R_{h-2} and so on (even if the different classes would be allowed to have different track sequences), leading to the conclusion that at most R_h codes are possible.

5 Limited track length and presorted trains

In this section, we consider other $\mathcal{NP}$-hard variants of the shunting problem.

Consider the situation of many independent trains that need to be rearranged. We assume that there is no restriction on the number of available tracks, but the capacity is limited. Independent here means that there is a one to one correspondence between incoming and outgoing trains, and a car goes only into the outgoing train corresponding to its incoming train. Accordingly, the order in which the incoming trains are pulled is irrelevant, the order on the track between cars from different trains is not important. The only interaction between the shunting processes for the different trains is the shared capacity constraint of the tracks.

In this setting, a train is completely specified by the lengths of its runs. Because there is a dedicated track for the outgoing train, the first (bottommost) car of the outgoing train is the last of the incoming train, and is the only car

that can be shunted directly. Now, the list of run-lengths specifies the runs of the outgoing train, and the incoming train has the same runs just in reverse order.

5.1 Strange setting, easy proof

Assume that the different tracks can have different capacities, and that the order in which the tracks are pulled is part of the input.

Problem "Shunting Independent Trains on Given Capacities with cars being pulled once"

Input A set of trains, specified by the lengths of the runs (i.e. an ordered list of numbers per train);
A list of capacities of tracks.

Output Is it possible to shunt all the trains using the tracks of the specified capacities in their given order, with the constraint that every car can participate in only one track pull.

Theorem 3. *"Shunting Independent Trains on Given Capacities with cars being pulled once" is $\mathcal{NP}$-hard.*

Proof. By reduction from "Not all equal 3-SAT", which is known to be $\mathcal{NP}$-hard [4, LO3]. We take an instance $\mathcal{S}$ of this problem and transform it, using the following observations. In our setting, no two cars of different runs of the same train may be planned to visit the same track. We choose that all the M trains (as described later) have precisely one run less than there are track pulls. Hence, for every train there is only one local decision, namely which of the runs to unnecessarily split (and how), or between which runs to put an unused track. Almost all runs of the trains consist of a single car, and most tracks have the according capacity M.

For every variable of $\mathcal{S}$, we have one train and enforce that the mentioned split run is either at the beginning or at the end. Then, the value of the variable is represented by the middle part of the train being "left" or "right". Now, it is easy to code a "not all equal" clause. This requires two neighboring specific tracks of capacity $M + 2k$, and runs of length $k + 1$ at the corresponding position in the trains representing the variables of the clause. Now, if the clause is satisfied the capacity requirement is $M + 2k, M + k$ or $M + k, M + 2k$ which is feasible. Otherwise the capacity requirement is $M + 3k, M$ or $M, M + 3k$ which is infeasible.

To make sure that the whole middle part of a train does not contain the split run, we employ the following construction, introducing a second train. Both have at a specific beginning position a run of length $k \geq 2$, and at a specific ending position a run of length $k + 1$. For both there is a window of two neighboring tracks that have capacity $M + k$. This fits easily if one of the trains does not use the first of the beginning tracks, and the other does not use the second of the ending tracks, and there is no split run or unused track in the middle. Now it is impossible that one of the trains has the split run or the unused track in

the middle: In this case, it would necessarily occupy all of the capacity at the first track at the beginning, and the second track at the end. But then the other train would need to have both of these tracks left empty, which is impossible.

In the concrete setting, it is sufficient to choose $k = 2$, i.e., one additional car in the described runs. Setting k to the number of used trains shows that keeping a strict bound on the car pulls makes it impossible to approximate the relative overload of a track to a factor better than 4/3.

5.2 Extensions

Here, we sketch a reduction that shows that even the problem of rearranging a single train on tracks of limited length is $\mathcal{NP}$-hard.

First, we want to replace the individual capacity constraints by a uniform one. To this end, we add one train that has precisely as many runs as there are track pulls. Now, because every car is only allowed to be pulled once, the shunting plan for this train is unique. By adjusting the lengths of the runs of this train, the differences in the capacity constraints can be adjusted.

To enforce that every car is pulled at most once, we add one train with one big non-trivial run. Assume the capacity constraint is c, the number of car pulls is h, and the total number of cars in the trains is n. We choose the length of this run to be $hc - n$. Now, if any car would be pulled twice another car could not be pulled at all, which is impossible in a correct shunting plan.

Finally, we can transform the setting of many trains into one of one train. Changing from many incoming trains to one incoming train is no problem, we can have the very same structure of runs. Changing to one outgoing train is not that straightforward because it would specify an order between the runs of the different trains, dictating the solution.

To this end, we make sure that again only codes with one one and two ones are allowed. It is convenient to choose fairly long tracks, and to have the single pull cars use most of the track and adjust the remaining height. This is possible as formulated in the following lemma, which leads to the claimed $\mathcal{NP}$-hardness result.

Lemma 2. *Given a set $\mathcal{S}$ of m independent trains, specified by a number of runs l_i, and the length of the runs a_{ij}, and a capacity constraint c, and a number of track pulls h. Assume further, that the total number of cars in $\mathcal{S}$ is hc, such that $\mathcal{S}$ can only be shunted with single car pulls, if at all. Then there is a polynomial time computable single train $\mathcal{T}$ and polynomial capacity constraint C and number of track pulls H, such that $\mathcal{T}$ can be shunted within the constraints if and only if the set of trains could be shunted within their constraints.*

5.3 2-approximation

In this section we again consider the situation of a single incoming train with n cars in a given order, that is shunted into a single outgoing train. The shunting yard has sufficiently many different tracks, but each track can only hold c cars.

Note that several incoming trains that are first shunted in a particular order are equivalent to a single incoming train.

Assume we know the optimal number h of track pulls, for example by trying all possible values for $h \leq n$. Then the number of car pulls is bounded by $m = c \cdot h$. Now assume that we can find a code that uses at most this number of ones in total, and only h positions (this is quite different from the capacity constraint). Now, we transform this into a code that obeys the capacity constraint and has $2h$ positions. If position i is overfull, say it has $k > c$ ones, we split this position into $d = \lceil k/c \rceil$ positions in the following way: We split all codes at position i, insert $d-1$ zeroes, and continue with the code, effectively moving the positions $> i$ by $d-1$ to the right. We take the k codes that have a one at position i and arbitrarily split it into d classes, the first $d-1$ having c codes, and the last class with fewer codes. The codes in the first class remain unchanged, the codes in the second class change the one in position i into a one in position $i+1$, and so on. None of the positions $i, \ldots, i+d-1$ have more than c codes. We repeat this procedure for all positions that have more than c codes. At the end, there at most h full tracks (because otherwise there would be more than $c \cdot h$ ones in total), and at most h not-full tracks (because every position of the original code with h positions can lead to at most one position with less than c ones), yielding a 2-approximation.

This leaves us with the task of deciding if it is possible to bound the car pulls by m and the track pulls h. Let the number of cars in the train be n.

Let us consider the following optimization problem: what is the minimal number of car pulls necessary to shunt the train with h track pulls? We observe that there are at most $\binom{n}{2}$ runs possible, the intervals of the outgoing train. Denote for a run X the number of cars in that run by $|X|$. Now, we generalize the question to "what is the minimal number of car pulls necessary to create run X on track i?" Every track pull unites only pairs of runs, so that X must stem from two sub-runs X_1, X_2, and X_1 is created on track i with m_1 car pulls, whereas X_2 is created on track $i' < i$ with m_2 car pulls, and the number of car pulls is $m_1 + m_2 + |X_2|$. This immediately suggests a dynamic programming approach, where the tabel is indexed by 'run', 'track', and the entry is 'minimal car pulls', the table size is $O(n^2 h)$. We allow the entry $+\infty$ and fill this table track by track, starting from the small (number of cars) runs. To fill the table for run X on track i we have to minimize over the $i-1$ choices for track i', and the $|X|$ choices of splitting the runs. This dynamic programming runs in polynomial time, namely $O(n^3 h^2)$. By inspecting the table for the complete run on the different tracks, it is easy to find all h that allows the complete run/train with at most $h \cdot c$ car pulls. Because of $h \leq n$, this dynamic programming runs in $O(n^5)$ time.

6 Acknowledgment

I would like thank Marc Nunkesser and Sebastian Stiller for inspiring discussions about the problem.

References

1. E. Dahlhaus, P. Horák, M. Miller, and J. F. Ryan. The train marshalling problem. *Discrete Applied Mathematics*, 103(1-3):41–54, 2000.
2. E. Dahlhaus, F. Manne, M. Miller, and J. Ryan. Algorithms for combinatorial problems related to train marshalling. In *Proceedings of AWOCA 2000, In Hunter Valley*, pages 7–16, July 2000.
3. R. Freling, R. Lentink, L. Kroon, and D. Huisman. Shunting of passenger train units in a railway station. *Transportation Science*, 39(2), 2005.
4. M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, 1979.
5. M. Gatto, R. Jacob, and M. Nunkesser. Optimization of a railway hub-and-spoke system: Routing and shunting. In *Poster Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms (WEA05)*. CTI Press, 2005.
6. M. E. Lübbecke and U. T. Zimmermann. Shunting minimal rail car allocation. *Comput. Optim. Appl.*, 31(3):295–308, 2005.