Train Marshalling Problem
- Algorithms and Bounds -

Katharina Beygang* Florian Dahms† Sven O. Krumke*

Abstract

The Train Marshalling Problem consists of rearranging an incoming train in a marshalling yard in such a way that cars with the same destinations appear consecutively in the final train and the number of needed sorting tracks is minimized. Besides an initial roll-in operation, just one pull-out operation is allowed. This problem was introduced by Dahlhaus et al [2] who also showed that the problem is $\mathcal{NP}$-complete.

In this paper, we provide a new lower bound on the optimal objective value by partitioning an appropriate interval graph. Furthermore, we consider the corresponding online problem, for which we provide upper and lower bounds on the competitiveness and a corresponding optimal deterministic online algorithm.

We provide an experimental evaluation of our lower bound and algorithm which shows the practical tightness of the results.

Keywords: Train Rearrangement, Online Algorithms, Competitive Analysis, Greedy Heuristic, Bell Number

1 Introduction

Marshalling yards (hump yards) play a decisive role in railroad life. They are responsible for arranging freight cars into specific sequences to assemble specific trains. A shunting yard consists of a hump, a set of parallel classification tracks (sorting tracks) and a roll-in and pull-out track. Any car which arrives at the shunting yard, rolls down from the hump, via the roll-in track to a classification track. The pull-out track reunites the cars resp. the block of cars by placing all cars from one of the tracks at the beginning of the rearranged train, followed by the cars of another track and so on such that all cars with the same destination are blocked together.

In such a marshalling yard, the most elementary operations are

*Department of Mathematics, University of Kaiserslautern, Paul-Ehrlich-Str. 14, 67663 Kaiserslautern, Germany. {beygang,krumke}@mathematik.uni-kl.de

†Department of Operation Research, RWTH Aachen University, Templergraben 64, 52062 Aachen, Germany. florian.dahms@googlemail.com

- decoupling the trains in front of the hump and pulling them over the hump (roll-in operations), and
- pulling the cars out of the classification tracks and reuniting them at the pull-out track (pull-out operations).

Both processes are operated by shunting locomotives. Due to limited resources and reasons of economy, the most important parameters in rearranging trains (cars) are

- the number of classification tracks,
- the length of classification tracks,
- the number of roll-in operations, and
- the number of pull-out operations.

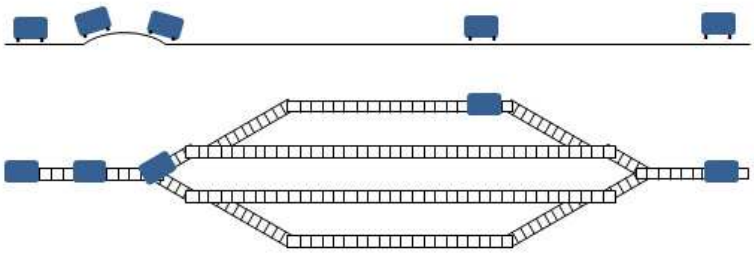

Figure 1 – Structure (side view and bird's eye view) of a marshalling yard.

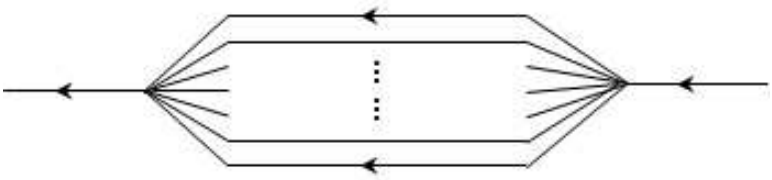

Figure 2 – Abstract illustration of a hump yard.

Figure 1 provides a schematic representation of a marshalling yard, shown during roll-in and pull-out operations. We will use an abstract illustration of a hump yard throughout this paper (see Figure 2).

In this paper we study the Train Marshalling Problem (TMP for short) where the car capacity of each classification track is set to be infinity. Additionally we allow one roll-in operation for every car and just one pull-out operation per track. This keeps the coupling and decoupling operations within a limit. Since it might be better that cars of a certain property, in particular with the same destination, appear consecutively (due to timetable restrictions etc.), we consider the problem of creating an outgoing train from a set of arriving trains where the cars with

certain properties are grouped together, i.e., form a block. Thereby, we do not require that the blocks have to appear in a certain ordering.

The paper is organized as follows: after the problem definition (Section 1.1), some preliminaries (Section 1.2) and an overview of related work (Section 1.3), we discuss the offline scenario in Section 2. We present some basic results and lower bounds on the optimal solutions. In Section 3, the online version of the Train Marshalling Problem is analyzed. We develop a deterministic online algorithm using greedy strategies. Afterwards we show that its competitive factor of 2 is in fact best possible among deterministic algorithms. In Section 4 computational results for the online algorithm and lower bounds are presented. We conclude with an appendix providing the theoretical background on how to generate instances for the computational experiments.

1.1 Problem Definition

Each *car* i arriving at the hump is identified with a positive integer σ_i denoting its destination. A *train* σ of length n is defined as a sequence of cars, i.e. $\sigma = (\sigma_1, \ldots, \sigma_n)$ with $\sigma_i \in \mathbb{N}, i \in \{1, \ldots, n\}$. The objective is to find the minimum number of movements of the shunting locomotives for the pull-out operations which are necessary to rearrange the train according to the destinations. This is equivalent to the target of minimizing the number of classification tracks which we denote by $K(\sigma)$.

Example 1.1. *Sequence* $\sigma = (1, 2, 2, 1)$ *with* $\sigma_1 = \sigma_4$ *and* $\sigma_2 = \sigma_3$ *represents a train whose first and last car have destination* 1*, while the second and third car have to go to destination* 2*.*

It will sometimes be convenient to reformulate the problem as Dahlhaus et al. [2] have done: Consider the set $I_n = \{1, 2, \ldots .n\}$ where every element stands for a car of a train of length n, and a partition of the set $S = \{S(1), \ldots, S(t)\}$ whose elements correspond to the destinations. The latter means that an element of S, i.e., $S(k)$, contains all cars having destination k.

Example 1.2. *The equivalent formulation of instance* σ *from Example 1.1 is* $S = \{S(1), S(2)\}$ *with* $S(1) = \{1, 4\}$ *and* $S(2) = \{2, 3\}$*.*

Thus,TMP reads as follows:

Definition 1.1 (TMP)**.** Find the smallest number $K(S)$ so that there exists a permutation $\pi(1), \ldots, \pi(t)$ of $1, \ldots, t$ so that the sequence of numbers

$$1, 2, \ldots, n, 1, 2, \ldots, n, 1, 2, \ldots, n$$

where the sequence $1, 2, \ldots, n$ is repeated $K(S)$ times contains all the elements of $S(\pi(1))$, followed by the elements of $S(\pi(2))$ and so on.

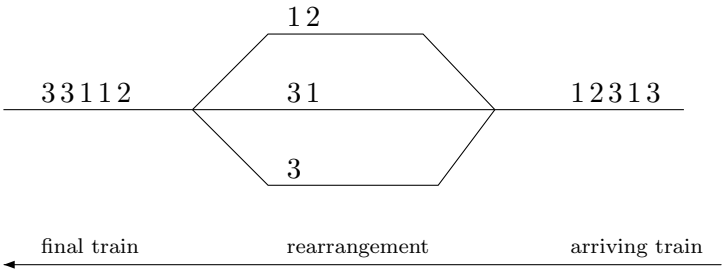

Figure 3 – Example of rearranging train $S = \{S(1), S(2), S(3)\}$ with $S(1) = \{1, 4\}$, $S(2) = \{2\}$ and $S(3) = \{3, 5\}$.

After a thought of moment, we see that both formulations are equivalent, in particular $K(S) = K(\sigma)$. For convenience, in Section 2 we will use the formulation stated in Definition 1.1. In Section 3, the instances will be defined as sequences. In general, if the input is denoted by S, we are talking about the representation via sets. Otherwise, the formulation via sequences is used.
In the following, "element of S" and "number", "destination" and "car" as well as "round" and "track" denote the same.

1.2 Preliminaries

Here, we provide the reader with the necessary notations needed in the following sections. Therefore, we will use TMP instances S defined via sets (formulation of Dahlhaus and others). It should be obvious how one can define everything equivalently in terms of input sequences σ.

Let $\mathbb{S}^n$ be the *set of all problem instances* of TMP with n cars. For $S \in \mathbb{S}^n$, the *number* of destinations is denoted by $d(S)$. In the following we abbreviate $d(S)$ by d if it does not cause confusions.

A *feasible rearrangement* for an instance $S \in \mathbb{S}^n$ is a mapping $\mathrm{tr} : \{1, \ldots, n\} \to \{1, \ldots, d\}$ using $|\,\mathrm{tr}\,|$ tracks. Note that we can always find a feasible rearrangement by simply assigning each destination on its own track. This results in $K(S) \leq d$.

The *first* and *last car* in S of destination k (with $1 \leq k \leq d$) is denoted by $first(S, k)$ and $last(S, k)$, respectively. Furthermore we denote the *first* and *last car of a track* i as $first(i)$ and $last(i)$, respectively.

Now let $\mathrm{tr} : \{1, \ldots, n\} \to \{1, \ldots, d\}$ is a rearrangement of S. Then, a destination is called *split* if two cars with the same destination have been assigned to two different (classification) tracks, i.e., there exist $i, j \in \{1, \ldots, n\}$ with $i, j \in S(k)$ (and $1 \leq k \leq d$) and $\mathrm{tr}(i) \neq \mathrm{tr}(j)$. Otherwise we are talking about an *unsplit destination*.

It will be helpful to distinguish between *complete* and *incomplete destinations*. Assume that we have already assigned step by step the first k cars of S to the tracks. Then a destination l is called *complete*, if all cars with destination l have already been assigned, i.e., for all

$1 \leq i \leq n$ with $i \in S(l)$ we have $i \leq k$. Otherwise we call a destination *incomplete*.

For instances denoted by sequences, let Σ^n be the set of all problem instances of length n. Analogously, for $\sigma \in \Sigma^n$ we can define $d(\sigma)$, $first(\sigma, k)$, and $last(\sigma, k)$ and so forth.

1.3 Related Work

In this paper, our focus will be on the Train Marshalling Problem allowing one sorting step while leaving freedom to the ordering of the destinations, in particular of the cars. It is first investigated and analyzed by Dahlhaus and others [2]. They show that the decision variant of this problem is, in general, $\mathcal{NP}$-complete by providing a reduction from Numerical Matching with Target Sums which is also known to be $\mathcal{NP}$-complete [7]. Furthermore the authors prove that for any instance at most $\lceil \frac{n}{4} + \frac{1}{2} \rceil$ tracks are needed where n denotes the number of cars of the inbound train. In [3], Dahlhaus and others discuss the problem for the case that the final position of each car is neither fixed nor arbitrary but has to fulfill certain requirements. To specify the ordering, they use a P-Q tree where the leaves are the cars and the inner nodes correspond to "blocks", i.e., groups of cars. In blocks which are marked as P-nodes, the sub-blocks, in particular cars, can be permuted in any order in the final train. In blocks which are Q-nodes, the sequence of immediate sub-blocks is predetermined. Knowing that this kind of problem is $\mathcal{NP}$-complete, they investigate special cases and provide approximate solutions.

There is abundant literature in the field of railway optimization. A detailed survey about other commonly used and new shunting, marshalling or classification problems is given by Gatto and others [8]. They approach the problems from an algorithmic point of view to give an entry point in the field of railway optimization. Di Stefano and others proposed models for rearranging cars of trains in [5] as well. It is intended for the project ARRIVAL - "Algorithms for Robust and online Railway optimization: Improving the Validity and reliAbility of Large scale systems".
The problem of sorting a sequence of numbers using a network of queues and stacks is presented in [6] and [13]. Di Stefano and Koči [4] deal with the problem of how to assign the arriving tram to the night depot such that they can be pulled out with a minimal number of shunting operations in the next morning.
Winter and Zimmermann [14] consider daily dispatch problems of trams in storage yards. Immediately on arrival, each tram has to be assigned to a location in the depot and possibly to an appropriate round trip of the next schedule period. Thereby, the dispatcher has to take into account that different round trips may require different types of trams. Winter and Zimmermann present binary program models for minimizing the amount of shunting and for minimizing the number of type mismatches, i.e., the number of round trips to which a tram of wrong

type is assigned to. Additionally they consider the computational complexity of different dispatch problems.
Hansmann and Zimmermann [10] consider sorting problems in rolling stocks consisting of finding an optimal schedule for rearranging units of rolling stock at shunting yards featuring a hump.

2 The Offline Problem

In the following we will differentiate the input instances of TMP between overlapping and non-overlapping instances. Therefore, for $S \in \mathbb{S}^n$, let $I_k = [first(S,k), last(S,k)]$ with $k = 1, \ldots, d$ denote the real line interval that starts in $first(S,k)$ and ends in $last(S,k)$. Hence, each $S(i), k = 1, \ldots, d$ can be uniquely identified with an interval I_k. The family of all these intervals is denoted by $\mathcal{I}_S$.

Definition 2.1 (Interval graph associated with an input instance)**.** For a given instance S of TMP we denote by $G_S = (V, E)$ the *interval graph* w.r.t. $\mathcal{I}_S$, where

- $V = (I_1, \ldots, I_d)$,
- $I_k, I_j \in E$ with $1 \leq k, j \leq d$ if and only if $I_k \cap I_j \neq \emptyset$.

Definition 2.2. Let $S \in \mathbb{S}^n$. Then, two destinations $1 \leq i, j \leq d$ do not *overlap* if $last(S,i) < first(S,j)$ or $last(S,j) < first(S,i)$. Otherwise they do overlap. An instance S of TMP is called *overlapping* if all destinations do pairwise overlap. Otherwise it is called *non-overlapping*.

To check whether all destinations overlap needs $\mathcal{O}(n)$ time by computing a maximum size clique in the corresponding interval graph G_S (the sorting of the intervals with respect to their left endpoints can be accomplished in $\mathcal{O}(n)$ time since all values are in the range $\{1, \ldots, n\}$). If the instance is overlapping, the size of the maximum clique in G_S, and d, the number of destinations, coincide.
Similarly, if we take the view on the input as an input sequence $\sigma \in \Sigma^n$, we can define the corresponding interval graph G_σ analogously. Note that we can transform every TMP instance $S \in \mathbb{S}^n$ into an input sequence $\sigma \in \Sigma^n$ and vice versa in $\mathcal{O}(n)$ time.

Before we will give a first lower bound on the optimal solution $K(S)$ where $S \in \mathbb{S}^n$, we note basic properties of ceiling functions and state some obvious propositions.

Remark 2.1.

- *If $t \in \mathbb{N}$ is even, we have $\lceil \frac{t-1}{2} \rceil = \lceil \frac{t}{2} \rceil < \lceil \frac{t+1}{2} \rceil$.*
- *If $t \in \mathbb{N}$ is odd, we have $\lceil \frac{t-1}{2} \rceil < \lceil \frac{t}{2} \rceil = \lceil \frac{t+1}{2} \rceil$.*

Proposition 2.1. *Let $S \in \mathbb{S}^n$ and $S' = \{S(i_1), \ldots, S(i_r)\}$ with $1 \leq i_1, \ldots, i_r \leq d$ be an arbitrary subset of S. Then we have $K(S') \leq K(S)$.*

Proof. Obvious, because every rearrangement of S is a feasible rearrangement of S' as well. □

2.1 Lower Bounds on Optimal Solutions

Here, we will establish some lower bounds on the optimal objective value.

Theorem 2.1. *Let $S \in \mathbb{S}^n$ be an overlapping instance. Then, we have $K(S) \geq \lceil \frac{d+1}{2} \rceil$.*

Proof. Let $S \in \mathbb{S}^n$ and O be an arbitrary permutation of $1, \ldots, d$ which represents an arbitrary ordering of the destinations of S in the final train.
The assignment w.r.t. O of the cars to the tracks looks as follows: At the beginning of the first round we place all cars from destination $O(1)$. Then we place all cars of $O(2)$ beginning with car $i \in S(O(2))$ where $i > last(O(1))$ is smallest. Since the destinations do pairwise overlap, the second round contains some elements of destination $O(2)$ as well. Otherwise $S(O(1))$ and $S(O(2))$ would not overlap. Then it is $O(3)$'s turn. Now we have to do a case distinction: Either we are able to place all its cars on the second round or its last car goes to the beginning of the third round. But in both cases, after placing all cars of $O(3)$, we are in the same position as we have been placing all cars of $O(2)$. We proceed with the remaining destinations in an analogous way. Obviously, we are able to string at most three destinations per round together except for the first and last round where at most two different destinations might occur. Figure 4 provides an illustration of the above procedure.

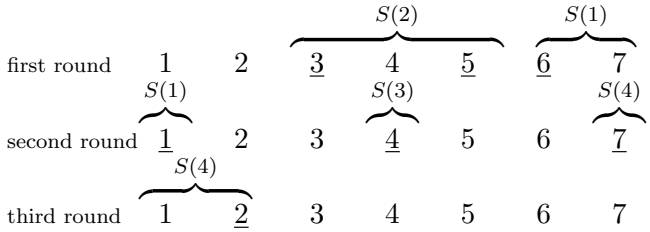

Figure 4 – Let S be an overlapping instance with $d = 4$ and $S(1) = \{1, 6\}$, $S(2) = \{3, 5\}$, $S(3) = \{4\}$ and $S(4) = \{2, 7\}$. If $O = (2, 1, 3, 4)$, the second round contains the first car of destination 1. Additionally the single element of $S(3)$ and the last car of destination 4 are placed in round 2.

We show by induction on $d = d(S)$ that if we are able to place exactly three destinations in one round by placing first the leftover of one destination $S(i)$, then a complete destination $S(j)$ and then the beginning of a third destination $S(k)$ for $i, j, k \in \{1, \ldots, d\}$ on it, then we have $K(S) = \lceil \frac{d+1}{2} \rceil$. Otherwise $K(S)$ becomes larger at best.
Now we suppose that O is such an ordering. Then we can assume that all destinations $S(O(k))$ with odd $O(k)$ are contained in exactly one round. All destinations $S(O(k))$ with even $O(k)$ are contained in two

rounds, i.e., its beginning at the end of one round and its end at the beginning of the consecutive one.
If $d = 1$, we need exactly one classification track because all cars of the train have the same destination. Thus we have $K(S) = 1 = \lceil \frac{d+1}{2} \rceil$.
Now we assume that $K(S) = \lceil \frac{d+1}{2} \rceil$ where $S = \{S(1), \ldots, S(d)\}$. Let S be a partition of I_n with $d+1$ destinations. By induction hypothesis we know that $\bar{S} = \{S(O(1)), \ldots, S(O(d))\}$ which is a subset of S together with order O needs exactly $\lceil \frac{d+1}{2} \rceil$ rounds. If $d+1$ is even then we know that we need another round to place them all. But then we have

$$\begin{aligned} K(S) = K(\bar{S}) + 1 &= \lceil \frac{d+1}{2} \rceil + 1 \\ &= \lceil \frac{d+1}{2} + 1 \rceil = \lceil \frac{d+3}{2} \rceil \\ &= \lceil \frac{d+2}{2} \rceil. \end{aligned}$$

The last equation is based on the fact that $d+1$ is even and Remark 2.1. For odd $d+1$ we know that no additional round is needed. Hence together with Remark 2.1 we have

$$K(S) = K(\bar{S}) = \lceil \frac{d+1}{2} \rceil = \lceil \frac{d+2}{2} \rceil.$$

□

Before we state a first lower bound on $K(S)$ for general instances $S \in \mathbb{S}^n$ we have to define a new parameter $\omega(S)$, called overlapping number, characterizing the state of the overlapping of S.

Definition 2.3. Let $S \in \mathbb{S}^n$ and G_S be the interval graph corresponding to $\mathcal{I}_S$. Then the *overlapping number* $\omega(S) \in \mathbb{N}$ is defined to be the size of a clique of maximum cardinality in the interval graph G_S associated with S.
Analogously, we denote by $\omega(\sigma)$ the size of a clique of maximum cardinality in G_σ, i.e., $\omega(\sigma) = \omega(S)$.

Theorem 2.2. *For $S \in \mathbb{S}^n$, we have*

- $\omega(S) \leq d$, *and*
- $K(S) \geq \lceil \frac{\omega(S)+1}{2} \rceil$.

Proof. The first part is obvious due to the fact that just d destinations can cause the overlapping number.
For the second part, choose a maximum clique of size $\omega(S)$ in G_S. Now, let S' be a partition obtained from S by deleting all destinations $S(k)$ which are not involved in this maximal overlapping of S, i.e., which are not contained in this maximum clique in G_S. This means, we reduce S to an overlapping instance $S' \in \mathbb{S}^m$ with $m \leq n$. Due to Proposition 2.1 and Theorem 2.1, we get

$$K(S) \geq K(S') \geq \lceil \frac{d(S')+1}{2} \rceil = \lceil \frac{\omega(S)+1}{2} \rceil.$$

□

Dahlhaus and others investigated an upper bound on the optimal solution which just depends on the number of cars. They proved in [2] that $K(S) \leq \lceil \frac{n}{4} + \frac{1}{2} \rceil$ with $S \in \mathbb{S}^n$. Together with Theorem 2.2, we get some optimality results for special instances:

Proposition 2.2. *Let $S \in \mathbb{S}^n$. If $\omega(S) = \lceil \frac{n}{2} \rceil$, we have $K(S) = \lceil \frac{\omega(S)+1}{2} \rceil = \lceil \frac{n}{4} + \frac{1}{2} \rceil$. In particular, this holds for overlapping instances with at most two cars per destination.*

Proof. Let S be such a required instance. Then the margin between upper and lower bound vanishes:

$$\begin{aligned} \lceil \frac{n}{4} + \frac{1}{2} \rceil - \lceil \frac{\omega(S)+1}{2} \rceil &\leq \lceil \frac{n}{4} + \frac{1}{2} - \frac{\omega(S)}{2} - \frac{1}{2} \rceil \\ &= \lceil \frac{n}{4} - \frac{\omega(S)}{2} \rceil \\ &= \lceil \frac{n - 2\omega(S)}{4} \rceil = 0. \end{aligned}$$

□

Now, let us establish a lower bound on the optimal solutions that dominates the bound $\lceil \frac{\omega(S)+1}{2} \rceil$ for $S \in \mathbb{S}^n$. Therefore, we have to define sub-instances S' of $S \in \mathbb{S}^n$ which result from S by removing elements of the destination sets:

Definition 2.4. Let $S \in \mathbb{S}^n$. Then $S' = \{S'(i_1), \ldots, S'(i_k)\}$ is called *sub-instance* of S if

- $i_s \neq i_t \ \forall s \neq t$, and
- $S'(i_k) \subseteq S(k)$.

In the following, we omit subsets of sub-instance S' which are empty.

Definition 2.5. Two sub-instances $D_1 = \{S'(i_1), \ldots, S'(i_{k_1})\}$ and $D_2 = \{S'(j_1)), \ldots, S'(j_{k_2})\}$ are *disjoint* if $i_{l_1} \neq j_{l_2}$ for all $l_1 \in \{1, \ldots, k_1\}$ and $l_2 \in \{1, \ldots, k_2\}$.

Additionally, we have to classify tracks in open and closed tracks by considering the destination of its last car. Therefore, we have to remember that a feasible assignment for $S \in \mathbb{S}^n$ is a mapping $\mathrm{tr} : \{1, \ldots, n\} \to \{1, \ldots, d\}$.
Assuming that the first $l \leq n$ cars are already assigned to tracks, we define a *partial assignment* by restricting the domain of tr to $\{1, \ldots, l\}$.

Definition 2.6. Let tr be a partial assignment of the first l cars and $i \in \mathrm{tr}(L)$ with $L = \{1, \ldots, l\}$. Then track i is called *closed*, if

- its last car has destination k, i.e., $last(i) \in S(k)$ and
- there exists another track $j \neq i$ with $j \in \mathrm{tr}(L)$ and $first(j) \in S(k)$.

Definition 2.6 comprises all tracks whose last car belongs to a split destination. The naming is based on the fact that we are not able to unite destination k to a block at the final train if we assign car $j > last(i)$ with another destination to closed track i.
To obtain a lower bound on the optimal solutions, we determine two disjoint sub-instances. Then, with the help of their overlapping numbers, we can state the following result:

Theorem 2.3. *Let* $S \in \mathbb{S}^n$ *and*

$$D_1 = \{S'(i_1), \ldots, S'(i_k)\} \text{ and } D_2 = \{S'(j_1), \ldots, S'(j_l)\}$$

two disjoint sub-instances, such that the last car of D_1 *is being sent before the first car of* D_2*. Then we have* $K(S) \geq \lceil \frac{\omega(D_1)+\omega(D_2)}{2} \rceil$.

Proof. Let $\omega(D_1) := k_1$ and $\omega(D_2) := k_2$. Let OPT be an optimal rearrangement of $S \in \mathbb{S}^n$ using $K(S)$ tracks. Additionally, we suppose that OPT has already used $a \leq K(S)$ tracks after the last car of D_1 has been assigned. But then, due to the Pigeonhole Principle we know that $k_1 - a$ destinations are split. Thus, $k_1 - a$ tracks are closed after assigning all cars of D_1, so that $K(S) - (k_1 - a)$ tracks can be used for the remaining sequence, in particular for assigning the cars of D_2.
Since $K(S) - a$ tracks are still empty, we can use them to split destinations which yields at most $K(S) - a$ split destinations.
So OPT needs at least $k_2 - (K(S) - a)$ tracks. Since $K(S) - k_1 + a$ are left, it results in

$$\begin{aligned} & k_2 - K(S) + a \leq K(S) - k_1 + a \\ \Rightarrow \quad & k_1 + k_2 \leq 2K(S) \\ \Rightarrow \quad & \frac{k_1 + k_2}{2} \leq K(S) \\ \Rightarrow \quad & \lceil \frac{k_1 + k_2}{2} \rceil \leq K(S). \end{aligned} \tag{1}$$

Inequality (1) holds because of the integrality of $K(S)$. □

To show that the condition of D_1 and D_2 being disjoint is crucial, consider instance

$$S = \{S(1), S(2), S(3), S(4), S(5)\} = \{\{1, 10\}, \{2, 5\}, \{3, 4\}, \{6, 9\}, \{7, 8\}\}$$

with

$$D_1 := \{S(1), S(2), S(3)\} \text{ and } D_2 := \{S(4), S(5)\}.$$

Note that $\omega(D_1) = 3$ and $\omega(D_2) = 2$. Applying the previous theorem, a lower bound on the necessary sorting track would be

$$K(S) \geq \lceil \frac{\omega(D_1) + \omega(D_2)}{2} \rceil = \lceil \frac{3 + 2}{2} \rceil = 3.$$

As can be seen easily, using permutation $\pi = (2, 4, 1, 3, 5)$ of the destinations, we just need two tracks for the rearrangement of S.

Now, consider $S = \{\{1,4\},\{2,7\},\{3,5,10,13\},\{6,9,12\},\{8,11\}\}$ with

$$D_1 = \{S(1), S(2)\} \text{ and } D_2 = \{S(3), S(4), S(5)\}.$$

By Theorem 2.3 we get $K(S) \geq 3$. Furthermore we have $\omega(S) = 3$ which results in $3 \leq K(S) \leq 3$. So S can be rearranged optimally using 3 tracks.

As the lower bound of Theorem 2.3 depends on the choice of the two sub-instances D_1 and D_2, we wish to find two sub-instances such that the resulting lower bound is maximized. Next we show that it is possible to find such optimal sub-instances within polynomial time.
Note that the disjointness property of D_1 and D_2 ensures that there must be a certain car c, such that D_1 ends before or at last with c and D_2 starts after c.
The idea is now to divide the problem instance $S \in \mathbb{S}^n$ into, not necessarily disjoint, sub-instances $D_1 = \{S'(1), \ldots, S'(d)\}$ and $D_2 = \{S'(1), \ldots, S'(d)\}$ at every possible position (car) $i = 1, \ldots, n$ in the following way: Let $i \in \{1, \ldots, n\}$. For all $k = 1, \ldots, d$ we have

$$c \in S'(k) \subset D_1 \Leftrightarrow c \leq i \text{ and } c \in S(k), \tag{2}$$
$$c \in S'(k) \subset D_2 \Leftrightarrow c > i \text{ and } c \in S(k). \tag{3}$$

Then, we can list the set of inclusionwise maximal cliques of G_{D_1} and G_{D_2}, denoted by M_1 and M_2. Remember that a clique of cardinality z corresponds to a set of destinations whose overlapping number is z. Since we are just interested in maximizing $\omega(D_1) + \omega(D_2)$ where any destination may only appear in one of the sub-instances, we have to find the pair $(m_1, m_2) \in M_1 \times M_2$ that maximizes $\omega(m_1) + \omega(m_2) - \omega(m_1 \cap m_2)$.
For each interval graph we can list all maximum cliques in $\mathcal{O}(n^2)$ time [9]. Additionally we have to run through $\mathcal{O}(n)$ positions (cars), each time comparing $\mathcal{O}(n^2)$ pairs of maximum cliques. Calculating the number of shared destinations for each pair of maximal cliques can be done in $\mathcal{O}(n)$ time. This yields a total running time of $\mathcal{O}(n^5)$ for the CLIQUE BOUND algorithm.

Algorithm 1 CLIQUE BOUND

Require: $S \in \mathbb{S}^n$
Ensure: lower bound l
1: $l = 0$
2: **for** $i = 1 \dots n$ **do**
3: Divide S into D_1 and D_2 at position i due to (2) and (3)
4: Find all maximal cliques in G_{D_1} and G_{D_2} and store them in M_1 and M_2, respectively.
5: **for** $(m_1, m_2) \in M_1 \times M_2$ **do**
6: $l = \max(l, \omega(m_1) + \omega(m_2) - \omega(m_1 \cap m_2))$
7: **end for**
8: **end for**
9: **return** l

3 The Online Scenario

Information about the number of cars arriving at the yard and their destinations, is the most important component for all computational tasks. Now we focus on the online version of TMP where the input is given piece by piece and is thus not available from scratch. This turns out to be a more realistic demonstration of railway life since interaction between all participants is proved to be difficult.
Hence, we consider the problem of computing a rearrangement of the freight cars with incomplete information and try to find a good online algorithm. To compare different online algorithms we need some kind of performance measure. In the field of online computation the method of competitive analysis, proposed by Borodin and El-Yaniv [1], has been established as one of the most successful ways to analyze the quality of an online algorithm. In competitive analysis we compare the results of the online algorithms with the optimal offline results, i.e., the best possible solution if we know all information in advance.

Definition 3.1. A deterministic online algorithm ALG is said to be *c-competitive*, if for every instance σ we have

$$\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) + \alpha.$$

The *competitivity* of ALG is the minimum over all c, such that ALG is c-competitive.

Our goal will be to provide competitive algorithms with good competitivity. But before, we want to show the necessity of the information if an arriving car is the last car of its destination or not.

Theorem 3.1. *There is no competitive deterministic online algorithm if the online algorithm does not have any information whether a car is a last car of its destination or not.*

Proof. Assuming the opposite, let ALG and OPT be a c-competitive and optimal algorithm for TMP, respectively. Note that we do not have any "last car" information. Then there exists a constant α with

$$\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) + \alpha.$$

Consider $\sigma \in \Sigma^n$ of the form $\sigma = (1, 2, \ldots, n)$. Obviously $K(\sigma) = 1$ and due to assumption $\text{ALG}(\sigma) \leq c \cdot 1 + \alpha$.

Now, consider $\sigma \in \Sigma^{2n}$ with $\sigma = (1, 2, \ldots, n, 1, 2, \ldots, n)$. Then ALG needs for the first n cars less than $c + \alpha$ tracks. But if $n > 2(c + \alpha)$, there exists at least one track where an incomplete destination is positioned between two other destinations. Since we allow just one pull-out operation, the final rearrangement will be infeasible. Hence, the claim follows. □

Greedy strategies are interesting because they are typically fast and use a small amount of memory. Now, let us consider an elementary greedy algorithm for the Train Marshalling Problem, denoted by GREEDY. The main idea is to assign the arriving car c to a track whose last car has the same destination as c has. If there is no track with this property, we send c to a track whose last car has a destination which is already complete. If no such track exists, we must open a new sorting track.

Algorithm 2 GREEDY

Require: $\sigma \in \Sigma^n$
Ensure: feasible assignment tr of σ

```
1: ω = 0 {overlapping number}
2: tempω = 0 {temporary overlapping number}
3: C = ∅ {set of available colors}
4: for i = 1...n do
5:     if i = first(σ, σ_i) then
6:         tempω++
7:         if u < tempω then
8:             ω++
9:             C = C ∪ {ω}
10:        end if
11:        tr(i) = c with c ∈ C
12:        C = C \ {c}
13:    end if
14:    if i = last(σ, σ_i) then
15:        tempω−−
16:        C = C ∪ {tr(i)}
17:    end if
18:    if i ≠ first(σ, σ_i) and i ≠ last(σ, σ_i) then
19:        tr(i) = tr(j) with i > j and σ_i = σ_j
20:    end if
21: end for
22: return tr : {1,...,n} → {1,...,ω}
```

Note that we can find an optimal coloring of the interval graph G_σ in linear time by using a greedy algorithm which considers the vertices in sequence and assigns each vertex its first available color [11].
Graph coloring assigns two different colors to adjacent vertices. Since these two vertices correspond to two overlapping destinations, they have to be assigned to different tracks. Thus, we get the following Corollary:

Corollary 3.2. *Every solution of* TMP *where we prohibit to split destinations and the solutions of the graph coloring problem in the corresponding interval graphs coincide.*

Corollary 3.3. GREEDY *outputs exactly* $\omega(\sigma)$ *tracks for* $\sigma \in \Sigma^n$.

Theorem 3.4. GREEDY *has a competitive ratio of at most* 2.

Proof. Let $\sigma \in \Sigma^n$. Then we get

$$\begin{aligned}\frac{\text{GREEDY}(\sigma)}{\text{OPT}(\sigma)} &\leq \frac{\omega(S)}{\lceil\frac{\omega(\sigma)+1}{2}\rceil} \\ &\leq \frac{2\cdot\omega(\sigma)}{\omega(\sigma)+1} \\ &< \frac{2\cdot\omega(\sigma)}{\omega(\sigma)} = 2\end{aligned}$$

□

The analysis of the greedy-type heuristics for TMP is quite easy. Now, we will show that its competitiveness is in fact the best competitiveness we can achieve for deterministic online algorithms.

Our lower bound construction works inductively. Assuming that an online algorithm has already assigned the first n arriving cars, we need to define an extension of $\sigma \in \Sigma^n$ by the next arriving car σ_{n+1}.

Definition 3.2. $\sigma' \in \Sigma^{n+1}$ is called an *extension* of $\sigma \in \Sigma^n$ by car $\sigma_{n+1} \in \mathbb{N}$ if σ is a subsequence of σ' that can be derived from σ' by deleting the last element and without changing the order of any element.
Analogously, for $S \in \mathbb{S}^n$, either we add $n+1$ to one of the already existing destination sets $S(1), \ldots, S(d)$ (car $n+1$ is not the first car with this destination) or enlarge the number of destinations by adding $S(d+1) = \{n+1\}$ (car $n+1$ is the first car of its destination).

Let ALG be an arbitrary deterministic online algorithm which outputs an assignment of σ. Then, we denote by $\text{ALG}(\sigma)$ the *set of all used tracks* for rearranging σ. The *number of closed tracks* of the problem solution $\text{ALG}(\sigma)$ is denoted by $C(\text{ALG}(\sigma))$. Furthermore, $D^2(\text{ALG}(\sigma))$ states the *number of tracks where two cars are already assigned to.*

Now, we will construct an instance with at most two cars per destination piece by piece based on the assignment decisions we made so far. At each step of this construction, we make sure that

(1) $|T| \leq 2 \;\forall\; T \in \text{ALG}(\sigma)$,
(2) $T \in C(\text{ALG}(\sigma))$ if $last(T)$ is the second car of destination $\sigma_{last(T)}$.

In order to simplify and shorten the notation, we combine both types of input instances. That means, if we want to express, how many cars of a certain destination σ_i have already been sent, we use $|S(\sigma_i)|$. Then the above conditions read as follows:

(1) $|T| \leq 2 \;\forall\; T \in \text{ALG}(\sigma)$,
(2) $T \in C(\text{ALG}(\sigma))$ if $|S(\sigma_{last(T)})| = 2$. (4)

Note that σ stands for the sequence of cars arrived so far. That means that there are at most two cars per track and any track that ends

with a car of a destination where both cars are already sent, is closed. Obviously, this implies that two cars on the same track have to vary in destinations. Otherwise, this track would not be closed. Furthermore, we will make sure that in any step of the instance construction either one of the two following conditions hold:

$$C(\text{ALG}(\sigma)) = D^2(\text{ALG}(\sigma)) \tag{5}$$

or

$$\begin{aligned}&(1)\ \exists\, T \in \text{ALG}(\sigma) : \text{ALG}(i) = T = \text{ALG}(j), i = first(T), |S(\sigma_i)| = 1\\ &(2)\ D^2(\text{ALG}(\sigma)) = C(\text{ALG}(\sigma)) + 1.\end{aligned} \tag{6}$$

Item (1) in condition (6) states that there exists a track with two cars and which first car is of a destination whose second car is not sent yet.

Lemma 3.5. *Let* ALG *be a deterministic online algorithm and* $\sigma \in \Sigma^n$ *be an instance satisfying conditions (4) and (6). Then there exists an extension* σ' *also satisfying condition (4) and either condition (5) or (6). Furthermore,* σ' *will satisfy* $C(\text{ALG}(\sigma')) = C(\text{ALG}(\sigma)) + 1$.

Proof. Let T be the track given by point 1 in condition (6) with $\sigma_{first(T)} = k$. Now extend σ to σ' by sending car $n+1$ with $\sigma_{n+1} = k$. It is not possible that ALG assigns car $n+1$ to track T due to the resulting infeasible rearrangement. Thus we know $\text{ALG}(n+1) = T'$ with $T' \neq T$. Then T' must be closed due to definition and we get $C(\text{ALG}(\sigma')) = C(\text{ALG}(\sigma)) + 1$. Additionally we know that $|T| \leq 2$ due to the argument above. Hence we have verified condition (4).
To satisfy one of conditions (5) or (6), we have to make a case distinction:

Case 1: $D^2(\text{ALG}(\sigma')) = D^2(\text{ALG}(\sigma))$, i.e., $n+1$ is the only car on track T'. Since condition (6) holds for σ, we have

$$\begin{aligned}D^2(\text{ALG}(\sigma')) &= D^2(\text{ALG}(\sigma))\\ &= C(\text{ALG}(\sigma)) + 1\\ &= C(\text{ALG}(\sigma')),\end{aligned}$$

which fulfills condition (5).

Case 2: $D^2(\text{ALG}(\sigma')) = D^2(\text{ALG}(\sigma))+1$, i.e., there are two cars assigned to track T'. But this implies

$$\begin{aligned}D^2(\text{ALG}(\sigma')) &= D^2(\text{ALG}(\sigma)) + 1\\ &= C(\text{ALG}(\sigma)) + 2\\ &= C(\text{ALG}(\sigma')) + 1.\end{aligned}$$

Hence it remains to verify item 1 of condition (6). Assuming the opposite, i.e., $i, n+1 \in \text{ALG}(T')$ with $|S(\sigma_i)| = 2$, T' would be a closed track due to conditions (4) before adding car $n+1$. Thus, assigning $n+1$ to track T' would be infeasible which completes the proof.
□

Lemma 3.6. *Let* ALG *be a deterministic online algorithm. Then for any* $d \in \mathbb{N}$ *there exists an instance* $\sigma \in \Sigma^k$ *with* $k \leq 2d$ *and* $d(\sigma) = d = |\text{ALG}(\sigma)|$.

Proof. Let d be the number of destinations of an instance of TMP. Then we will proceed by induction on d and show that for all $d \in \mathbb{N}$ there exists a sequence σ which satisfies conditions (4) and (5) (Induction Hypothesis).

For $d = 1$, the only possible instance is $\sigma = (\sigma_1, \sigma_2)$ with $\sigma_1 = \sigma_2$. Hence the hypothesis is true.

Let us now assume that the claim holds for $d \geq 2$. Thus, we can find a sequence σ_d with d destinations fulfilling both conditions.

Extending σ_d by car c with destination $d+1$ results in sequence σ_{d+1} with one more car and one new destination. Consider the track $T \in \text{ALG}(c)$ to which c is assigned.
Since $|S(\sigma_c)| = 1$ and all other tracks remain unaffected, item 2 of condition (4) is still valid and $T \notin C(\text{ALG}(\sigma_{d+1}))$, i.e., T is not a closed track. The solution of ALG is not feasible if there are 3 cars upon T since we already know that the first two cars have to be different in destinations. This completes condition (4).

It remains to show that σ_{d+1} fulfills condition (5). Since c is the first car of its destination, the number of closed tracks remains the same, i.e.,

$$C(\text{ALG}(\sigma_{d+1})) = C(\text{ALG}(\sigma_d)).$$

Now let us consider the following case distinction in D^2.

Case 1: $D^2(\text{ALG}(\sigma_{d+1})) = D^2(\text{ALG}(\sigma_d))$. Then, due to the induction hypothesis, we have

$$D^2(\text{ALG}(\sigma_{d+1})) = D^2(\text{ALG}(\sigma_d)) = C(\text{ALG}(\sigma_d)) = C(\text{ALG}(\sigma_{d+1}))$$

and we are done with condition (5).

Case 2: $D^2(\text{ALG}(\sigma_{d+1})) = D^2(\text{ALG}(\sigma_d)) + 1$. Analogously, this yields

$$D^2(\text{ALG}(\sigma_{d+1})) = D^2(\text{ALG}(\sigma_d)) + 1 = C(\text{ALG}(\sigma_d)) + 1$$

and point 2 of condition (6) is verified.

Additionally, due to our assumptions, we know that there are two cars assigned to T with $first(T) = x$ and $last(T) = c$. Since T was not a closed track in $\text{ALG}(\sigma_d)$ (otherwise we had not been allowed to pull c upon T), we get

$$|S(\sigma_x)| = 1,$$

i.e., the second car of destination of car x has not been sent yet. This completes condition (6).

Then, by Lemma 3.5, we know that there exists an extension σ'_{d+1} which exhibits the same number of destinations by sending just cars with destinations whose first car is already sent (see proof). This means, applying this lemma, we have two possible outcomes. Either we find an enlarged instance which satisfies conditions (4) and (5) and we are done, or we find a sequence which fulfills conditions (4) and (6). But this allows us to apply Lemma 3.5 again. Hence, we recursively build an extension with constant number of destinations fulfilling condition (5). We know, that by applying this lemma and thus sending a second car of a destination, the number of closed tracks increases by one. Hence, we are just able to apply it while we have at most $d+1$ closed tracks since the instance exhibits just $d+1$ destinations. So, at the latest if we have build an instance with exactly two cars per destination, condition (5) is verified. This finishes the induction.

So, for all $d \in \mathbb{N}$ there exists an instance σ_d such that

$$D^2(\text{ALG}(\sigma_d)) = C(\text{ALG}(\sigma_d)). \tag{7}$$

This implies, that $d - C(\text{ALG}(\sigma_d))$ destinations are still incomplete, which yields

$$n = 2 \cdot C(\text{ALG}(\sigma_d)) + d - C(\text{ALG}(\sigma_d)) = C(\text{ALG}(\sigma_d)) + d \tag{8}$$

cars. Additionally we know that two cars are assigned to exactly $D^2(\text{ALG}(\sigma_d))$ tracks, which results due to equations (7) and (8) in

$$n - 2 \cdot D^2(\text{ALG}(\sigma_d)) = d - C(\text{ALG}(\sigma_d))$$

tracks with just one car. So, in total we get

$$|\text{ALG}(\sigma_d)| = D^2(\text{ALG}(\sigma_d)) + d - C(\text{ALG}(\sigma_d)) = d$$

tracks. □

Note that the proof does not necessarily find an instance with exactly two cars per destination.

Theorem 3.7. *There exists no deterministic online algorithm with competitive ratio smaller than* 2*.*

Proof. Let $d = 2k-1 \in \mathbb{N}$. Then by Lemma 3.6, we can find an instance σ_d with $d = |\text{ALG}(\sigma_d)|$. Since σ_d does not need to have exactly two cars per destination, we know that there exists an extension $\sigma \in \Sigma^{2d}$ with $|\text{ALG}(\sigma_d)| \leq |\text{ALG}(\sigma)|$.

Due to the upper bound on the optimal solution of Dahlhaus et al. [2], we also know

$$K(S) \leq \lceil \frac{2(2k-1)}{4} + \frac{1}{2} \rceil = \lceil \frac{4k-2}{4} + \frac{1}{2} \rceil = \lceil k - \frac{1}{2} + \frac{1}{2} \rceil = k.$$

Due to $\lim_{k\to\infty} \frac{2k-1}{k} = 2$, we get what we wanted to show.

□

4 Experimental Results

To get a feeling for the practical performance of the lower bounds and the online algorithm presented in this work, we implement them and analyze the performance for randomly generated instances.

We perform some computational experiments for 100 uniformly and independently chosen problem instances (for more information on this topic, see the Appendix).

To solve the Train Marshalling Problem to optimality, we developed a two-level procedure. The first level consists of relaxing requirements on feasible solutions of TMP. We solve the resulting integer program which turns out to be solvable in polynomial time for some special cases (see forthcoming PhD thesis of Katharina Beygang). The solution provides an upper and lower bound on the optimal solutions of the original problem. In some cases these bounds coincide, so that the optimal solution of TMP is found. Otherwise we run the second level taking advantage of these bounds.
In the following, l_C denotes the here presented lower bound on the optimal solution.

The relative error is defined as the absolute error (absolute value of the difference between optimal and computed value) divided by the optimal value. Here, it is expressed in percentage. The relative error of the lower bound l_C and heuristic GREEDY can be seen in Figure 5. Regardless of the number of cars, the relative error of l_C ranges consistently between 0 and 20% while the heuristic always yields a larger deviation of the observed output from the optimum. Thus, the performance of the lower bounds w.r.t. the relative error, is much better than GREEDY's.

This effect can also be seen in Table 1, considering the average deviation, i.e., the average of absolute errors. Here, the average and standard deviation for randomly generated instances, each of 50, 100 and 200 cars, is displayed in columns two to six, respectively. The standard deviation is a statistical value that shows how much variation there is from the average deviation. If the outputs are tightly bunched together, the standard deviation is small. If they are widely spread, the standard deviation is larger.
Here, observe the sharper concentration of l_C, simultaneously with the smaller average deviation, independent of the length n of the instances.

All computations were executed on a compute-server equipped with a Dual Intel Xeon 3.2GHz CPU and 4 GB RAM, running Linux Kernel 2.6.5 SMP. Programming framework for generating the instances was Perl, version 5.8.3. Lower bound l_C is implemented in Java Code compiled on JRE 1.5.0. Determining an optimal solution of TMP is done with solver suite IBM ILOG OPL 6.3, IBM ILOG CPLEX 12.1.0.

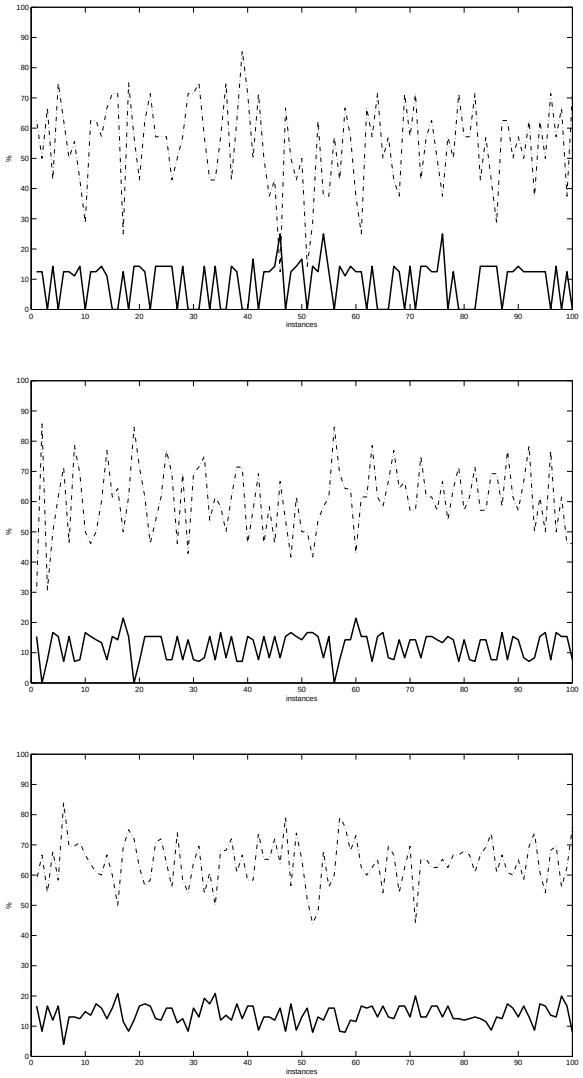

Figure 5 – Relative error ([%], y-axis) of GREEDY (dashed line) and l_C (dark line) for a sample of 100 randomly generated instances (x-axis) with $n = 50, 100$ and $n = 200$ cars (from top to bottom).

In Table 2, we present the maximum, minimum and average CPU run time of computing an optimal solution and lower bound l_C, as well as running GREEDY.
As expected, for increasing length n, the running time of GREEDY and the computation of l_C do not increase as much as the time for determining the optimal rearrangement.

	$n = 50$		$n = 100$		$n = 200$	
	d_{med}	σ	d_{med}	σ	d_{med}	σ
l_C	0.86	0.72	1.62	0.85	3.28	1.22
GREEDY	3.99	1.46	8.05	1.84	15.82	2.58

Table 1 – Average deviation d_{med} from optimum and standard deviation σ of introduced bound l_C and heuristic GREEDY.

	n	50	100	150	200
min	l_C	0.12	0.15	0.17	0.19
	GREEDY	0.02	0.03	0.06	0.09
	OPT	0.08	0.23	0.51	1.09
max	l_C	0.14	0.18	0.21	0.32
	GREEDY	0.03	0.04	0.06	0.17
	OPT	0.87	8.12	80.93	3033.36
∅	l_C	0.13	0.17	0.18	0.21
	GREEDY	0.02	0.03	0.06	0.1
	OPT	0.35	3.40	28.69	174.19

Table 2 – Minimum, maximum and average CPU run time of computing the optimal solution, l_C and of running GREEDY for 100 randomly generated instances, each with n cars.

5 Conclusions

We have provided a new polynomial time computable lower bound for optimal solutions in the offline version of TMP. The bound is based on graph theoretical methods and searches for inclusionwise maximal cliques in interval graphs. Furthermore, we considered the online version of TMP and showed that, from the viewpoint of competitive analysis, the best deterministic algorithm is a greedy-type algorithm which outputs a rearrangement using at most twice s many sorting tracks as necessary. When comparing the lower bound and heuristic experimentally, we determined that the lower bound tends to have a smaller relative/ absolute error w.r.t. the optimal value $K(S)$ than the output of the greedy heuristic.

There are still several questions that remain non-highlighted or unanswered in this paper, i.e.,

- What is a lower bound on the competitiveness of randomized online algorithms? Is there any randomized online algorithm which is better than 2-competitive?
- Are there any problem instances and problem restrictions which make the problem solvable in polynomial time?
- How does the 2-level procedure for solving TMP look like?

References

[1] Alan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

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

[3] Elias Dahlhaus, Fredrik Manne, Mirka Miller, and Joe Ryan. Algorithms for combinatorial problems related to train marshalling. In *Proceedings of AWOCA 2000, Hunter Valley*, pages 7–16, 2000.

[4] Gabriele Di Stefano and Koči Magnus L. A graph theoretical approach to the shunting problem. *Theoretical Computer Science*, 92:16–33, 2004.

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

[6] Shimon Even and Alon Itai. Queues, stacks, and graphs. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations: International Symposium on the Theory of Machines and Computations*, pages 71–86. Academic Press, Inc., 1971.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[8] Michael Gatto, Jens Maue, Matús Mihalák, and Peter Widmayer. Shunting for dummies: An introductory algorithmic survey. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos D. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 310–337. Springer, 2009.

[9] Udaiprakash I. Gupta, Der T. Lee, and Joseph Y-T. Leung. Efficient Algorithms for Interval Graphs and Circular-Arc Graphs. *Networks*, 12(4):459–467, 1982.

[10] Ronny S. Hansmann and Uwe T. Zimmermann. Optimal sorting of rolling stock at hump yards. In Hans-Joachim Krebs and Willi Jäger, editors, *Mathematics, Key Technology for the Future*, pages 189–203. Springer Berlin Heidelberg, 2008.

[11] Sven O. Krumke and Hartmut Noltemeier. *Graphentheoretische Konzepte und Algorithmen*. Teubner Verlag, Mai 2005.

[12] Gian-Carlo Rota. The number of partitions of a set. *The American Mathematical Monthly*, 71(5):pp. 498–504, 1964.

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

[14] Thomas Winter, Uwe T. Zimmermann, Abteilung für Mathematische Optimierung, and TU Braunschweig. Real-time dispatch of trams in storage yards. *Annals of Operations Research*, 96:287–315, 2000.

A Appendix

To generate instances for the computational analysis, we use the discrete uniform distribution on Σ^n where n is fixed.
Additionally, let $(\Sigma^n, \mathcal{P}^{\Sigma^n}, P)$ be the probability space, where Σ^n is the domain, $(\Sigma^n, \mathcal{P}^{\Sigma^n})$ is a measurable space, $\mathcal{P}^{\Sigma^n}$ are all measurable subsets of Σ^n and P is the measure on $\mathcal{P}^{\Sigma^n}$ with

$$P(\Sigma^n) = 1 \text{ and } P(\sigma) = \frac{|\sigma|}{|\mathcal{P}^{\Sigma^n}|}.$$

Note that the discrete uniform distribution is a probability distribution for which the probability of occurrence is the same for all values of Σ^n.

We will calculate P recursively. Therefore, consider Σ^n with $n = 3$. Without loss of generality, we can assume that the first car has destination 1, the second destination is denoted by 2 and so on.

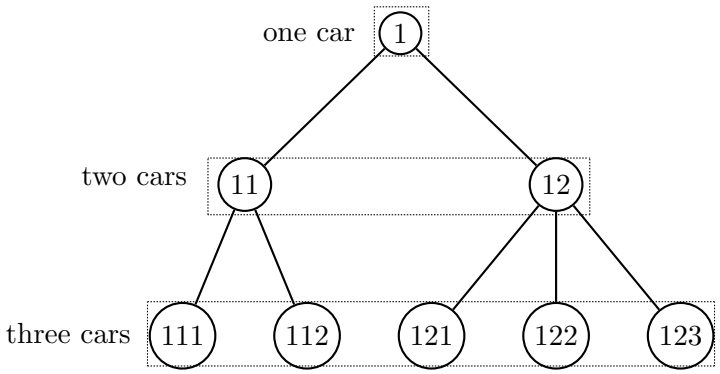

Figure 6 – Construction of Σ^3.

$\mathcal{P}^{\Sigma^3}$ contains 5 possible outcomes. Obviously, for each car in σ, we can either choose one of the destinations already sent (number of destinations stays the same) or we add a new destination.

Let $P(d, l)$ denote the number of instances where the first l cars use d destinations. Then, we conclude

Theorem A.1. *$P(d, l) = P(d+1, l+1) + d \cdot P(d, l+1)$ for $d \leq l$ and $P(d, l) = 1$ for $l \geq n$.*

Proof. For $l = n$, P is correct due to definition. Even the other part ($l < n$) is easy since we only have to consider the case of choosing a new destination $d+1$ or of choosing one of the d already used destinations. If we add the new destination, we are done with $P(d+1, l+1)$. If we choose one of the already used d destinations, we are left with $P(d, l+1)$ possibilities. Thus, we are done. □

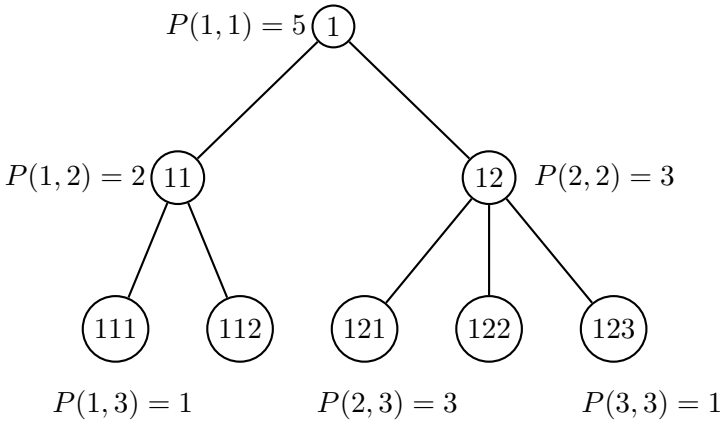

Figure 7 – Survey of $P(d, l)$ with $n = 3$.

Now, we are able to determine $P(d, l)$ for every $d \leq l$. As there are less than n^2 possible values for P, this can be done in $\mathcal{O}(n^2)$ time (Algorithm 3) by generating a table of all values.
Now we can use P to generate a uniform random sequence $(\sigma_1, \ldots, \sigma_n)$ car by car with $\sigma_1 = 1$. We just need to keep track of i and d denoting the current considered car and the number of already used destinations respectively, and look up the respective values of P we generated before. Algorithm 4 decides for each car i whether we attribute an already open (used) destination to i or not. In the latter case, i gets destination $d + 1$. The probability to choose an already used destination is calculated as $\frac{d \cdot P(d,l)}{P(d,l-1)}$, i.e., the number of possibilities we would have after assigning an "old" destination to car i divided by the total number of possibilities. In case we decide for one of the old destinations, we choose σ_i uniformly from $\{1, ..., d\}$. Note that once we have calculated P we can generate uniform random instances in linear time.

We can also look at these results in the following way: As every problem instance $S \in \mathbb{S}^n$ is a partition of the set $\{1, 2, ..., n\}$ and the procedure above can generate any partition of this set with equal probability, we have found a method of drawing uniformly a random partition of the set $\{1, ..., n\}$. Additionally, $P(1, 1)$ gives the total number of possible set partitions of a set of size n. Hence, it is equal to the n-th Bell number B_n [12].

Algorithm 3 P-CALCULATOR

Require: $n \in \mathbb{N}$
Ensure: $P(d, l)$ for all d, l
1: **for all** $l = n, n-1, \ldots, 1$ **do**
2: **for all** $d = 1, \ldots, l$ **do**
3: **if** $l < n$ **then**
4: $P(d, l) = P(d+1, l+1) + d \cdot P(d, l+1)$
5: **else**
6: $P(d, l) = 1$
7: **end if**
8: **end for**
9: **end for**
10: **return** $P(d, l)$ for all d, l

Pseudocode 4 uses the expression rand as a random number between 0 and 1 and uniform as the discrete uniform distribution over $\{1, \ldots, d\}$.

Algorithm 4 INSTANCE-GENERATOR

Require: $n \in \mathbb{N}$
Ensure: $\sigma = (\sigma_1, \ldots, \sigma_n)$
1: $d = 1\{\text{number of destinations}\}$
2: $\sigma_1 = 1$
3: **for all** $i = 2, 3, \ldots, n$ **do**
4: **if** rand $< \frac{d \cdot P(d,i)}{P(d,i-1)}$ **then**
5: $\sigma_i = \text{uniform}(1, \ldots, d)$
6: **else**
7: $d = d + 1$
8: $\sigma_i = d$
9: **end if**
10: **end for**
11: **return** σ