TRAIN MARSHALLING PROBLEMS - ALGORITHMS AND COMPLEXITY

FLORIAN DAHMS

Diplomarbeit

Supervisor: Prof. Dr. Sven O. Krumke

December 2010

Florian Dahms: *Train Marshalling Problems - Algorithms and Complexity*, Diplomarbeit, © December 2010

ABSTRACT

The goal of train marshalling is to rearrange an incoming train in such a way, using a marshalling yard, that it fulfills certain properties when leaving the yard. In this thesis we will consider the train marshalling problem proposed by Dahlhaus et al. [5]. Here we want to sort an incoming train to blocks of the same destination using only one classification step and a minimal number of sorting tracks, while being able to choose the order of the destinations freely.

We will state some basic properties of this NP-hard problem and derive some upper and lower bounds on the optimal objective value. For these we will apply some interesting results from graph theory. Also we will analyze easily solvable problem instances and state algorithms to solve them in polynomial time.

Furthermore we will discuss the corresponding online problem, where we will provide an optimal deterministic online algorithm in terms of competitive analysis. Also we will look at some kind of average case performance of the algorithms and bounds using uniform random instances.

From the results obtained so far we will then propose further results for variations of the original problem, such as restricting the number of sorting tracks, introducing an auxiliary track or allowing multiple sorting steps.

Gebraucht der Zeit, sie geht so schnell von hinnen,
doch Ordnung lehrt euch Zeit gewinnen.

— Johann Wolfgang von Goethe

ACKNOWLEDGMENTS

It is a pleasure to thank all those how made this thesis possible.

First of all I owe my gratitude to my supervisor Prof. Dr. Sven Oliver Krumke as well as Katharina Beygang for their great guidance and support.

And I want to thank my parents, who made all this possible, for all their encouragement throughout the time.

CONTENTS

1 INTRODUCTION 1
- 1.1 Overview 1
- 1.2 Other train sorting problems 3
- 1.3 Basic definitions 3
  - 1.3.1 Problem instances and solutions 4
  - 1.3.2 Sub-instances 8
  - 1.3.3 Interval graphs and overlappings 10
  - 1.3.4 Algorithms 11

2 OFFLINE PROBLEM 13
- 2.1 IP Formulation 13
- 2.2 Bounds 15
- 2.3 Easy instances 23
  - 2.3.1 Pair instances with maximal overlapping destinations 23
  - 2.3.2 Instances with large disjoint sub-instances 25
  - 2.3.3 Fixed number of destinations 26
  - 2.3.4 Fixed overlappings 26

3 ONLINE PROBLEM 29
- 3.1 Introduction to online computation and competitive analysis 29
- 3.2 Lower bound for det. online algorithms 31
- 3.3 Algorithms 40
  - 3.3.1 Split 40
  - 3.3.2 Unsplit 42
- 3.4 Randomized online algorithms 43

4 COMPUTATIONAL RESULTS 47
- 4.1 Underlying Probability Space 47
- 4.2 Results 50

5 PROBLEM VARIATIONS 55
- 5.1 Fixed number of tracks 55
  - 5.1.1 Offline Problem 55
  - 5.1.2 Online Problem 56
- 5.2 Using an auxiliary track 57
- 5.3 Multiple sorting steps 59

6 CONCLUSIONS 63
- 6.1 Open questions 63

BIBLIOGRAPHY 65

LIST OF FIGURES

Figure 1 Schematic hump yard [11] 1
Figure 2 Schematic hump yard with exit track 2
Figure 3 Example of sorting an incoming train according to the Train Marshalling Problem (TMP) model 2
Figure 4 An example for an interval graph with the corresponding intervals 11
Figure 5 Example for Algorithm 2 17
Figure 6 Example for necessity of complete separation of D_1 and D_2 21
Figure 7 Example for Theorem 2.11 21
Figure 8 Example for the first four cars of the malicious sequence depending on the choices of the algorithm 39
Figure 9 Malicious problem instance for the Split algorithm 42
Figure 10 Possible choices for S_1 and S_2 45
Figure 11 Possibilities when generating a sequence in $\mathcal{S}^3$ 48
Figure 12 Comparing bound and algorithm results for 100 instances in $\mathcal{S}^{50}$ 53
Figure 13 Comparing bound and algorithm results for 100 instances in $\mathcal{S}^{100}$ 53
Figure 14 Schematic hump yard with auxiliary track of capacity m 58
Figure 15 Example for multiple sorting steps 61

LIST OF TABLES

Table 1 Comparision of bounds and algorithms for $n = 50$ 54
Table 2 Comparision of bounds and algorithms for $n = 100$ 54

LIST OF ALGORITHMS

1 Calculate a solution from a permutation of the destinations . . . 6

2 Algorithm for optimal destination coloring . . . 16

3 Determine optimal bound for Theorem 2.11 . . . 23

4 Algorithm to determine optimal solution for instances from Corollary 2.15 . . . 24

5 The Split algorithm . . . 41

6 The Unsplit algorithm . . . 43

LISTINGS

Listing 1 Matlab code to generate a table with values for the possibility function P 49

Listing 2 Matlab code to generate uniformly distributed random instances of length n 50

ACRONYMS

TMP Train Marshalling Problem

IP Integer Program

1 INTRODUCTION

1.1 OVERVIEW

When considering railway logistics an important problem that naturally arises is the problem of shunting or marshalling railway cars. This means we need to find a list of instructions how to rearrange railway cars in a classification yard so that they form new trains according to certain specifications. Of course this needs to be done efficiently to reduce the necessary amount of resources (like time, space, the number of couplings and decouplings, etc.). In the following we will construct a certain model for this problem and present the results we obtained in the analysis of this specific problem. But first we will give a general introduction into the functioning of shunting yards.

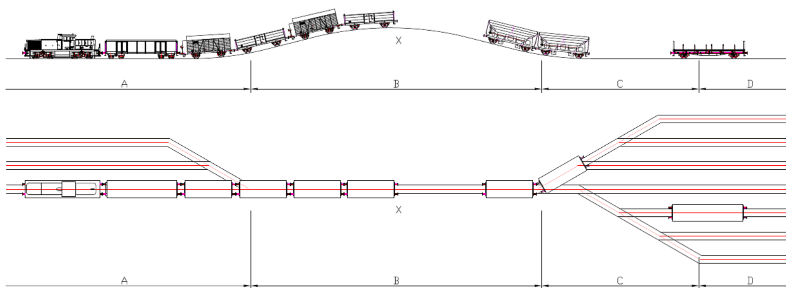


Figure 1: Schematic hump yard [11]

Shunting trains is usually done in hump yards

A very common form of classification yards are so called hump yards (a schematic hump yard can be seen in figure 1). The hump yard mainly consists of one incoming track that leads over a hill/hump (B), followed by several switches (C) and the classification tracks (D). Shunting the cars is now done by slowly pushing the decoupled cars over the hump. After a car crosses the tipping point it will roll down the hump by itself and is guided to its allocated track using the (mostly automated and remotely controlled) switches.

For illustration purposes we will also assume, that after the classification tracks we have one outbound exit track on which the cars will be pulled after they have been sorted (see figure 2).

In the European railway network the coupling and decoupling of cars is still done by hand as automatic hooks have not been introduced yet [6]. This makes coupling/decoupling a time consuming task that should be performed as little as possible. Bear-

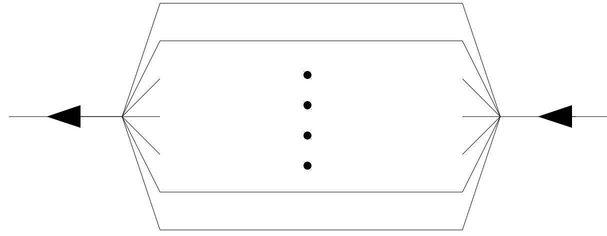


Figure 2: Schematic hump yard with exit track

We will allow only one humping step as couplings are time consuming

ing this in mind when constructing our model we will restrict ourselves to exactly one classification step. This way we only need to couple and decouple every car exactly one time.

Goal: Sort train by destinations with minimal sorting tracks

To make this more specific, the model will be to decouple all the cars on the incoming track, place them on their respective classification track, then couple all the cars on one classification track, pull out all cars from each track to the exit track where the outbound train will be joined together. The goal of the sorting process will be to arrive at a final train, where cars are sorted by destinations, i.e. cars that were assigned for the same destination appear in a connected order, while minimizing the number of necessary sorting tracks. In the following we will denote this model as TMP.

As the available space in the hump yard is a limited resource minimizing the necessary amount of it obviously makes sense, especially as the time components like shunting steps and number of couplings are already fixed to a be small.

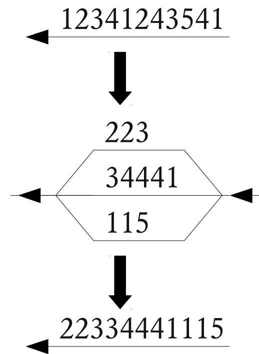


Figure 3: Example of sorting an incoming train according to the TMP model

In figure 3 we see an example of how such a sorting process might work. First we see a sequence of cars that are denoted by a number, referring to their respective destination. In the second step we see how they can be placed on 3 classification tracks so

that they will form a sorted train after joining the tracks in the third step.

From the example it is easy to see, that the assumption of having an outgoing track can easily be dropped. We then can simply pull out the train to the incoming track in reverse order, still having the final train sorted by destinations in the same way as before.

1.2 OTHER TRAIN SORTING PROBLEMS

Throughout this thesis our focus will be on the problem stated above, where we allow only one sorting step while leaving freedom to the ordering of the destinations and the cars within each destination. But there are also other models for different train shunting problems.

For example a very common model is to assume that an incoming train needs to be sorted in such a way, that each incoming car leaves at a certain, prefixed position of the outbound train. Again one considers objectives like the number of necessary sorting tracks and the number of shunting operations. Usually one allows several shunting operations like pulling out single cars and rolling them in again.

Two elementary sorting schemes in this model are the triangular and the geometric sorting scheme. For a more detailed overview about this topic the reader shall be referred to [7, 15].

Another class of shunting problems is related to the problem of tram scheduling. Here one considers a night depot where trams have to be placed in such a way, that they can be accessed optimally in the next morning.

One such model that deals solely with assigning tasks to different classes of trams was worked on in [2]. Here one considers a night depot, where trams of different types were already parked in a certain order. Now the question arises if it is possible to satisfy a certain sequence of tasks, each requiring a certain type of tram, while not changing the order of trams in the depot. This question is shown to be NP-hard, but Blasum et al. [2] propose a dynamic program that can solve real world instances.

Other models deal with the problem of how to assign the arriving trams to the night depot such that they can be pulled out without shunting operations in the next morning (see [16]).

1.3 BASIC DEFINITIONS

We will now make the definition of TMP more precise and provide the reader with the necessary notations needed in the following chapters. First we give the definition of a TMP instance and its basic properties.

1.3.1 *Problem instances and solutions*

Definition 1.1. We call a set

$$S = \{S_d, 1 \leqslant d \leqslant t\},\ t \in \mathbb{N}$$

of d sets that satisfy

$$i \in S_d \Rightarrow i \notin S_e \textit{ for } d \neq e$$

as well as

$$S_d \neq \emptyset \quad \forall\, 1 \leqslant d \leqslant t$$

a problem instance of TMP. That means S is a collection of t sets, where each set S_d contains the cars that will have to go to destination d (no car can be in two different sets, as it must be associated with exactly one destination).

$t(S)$ is the number of destinations

$n(S)$ is the number of cars

The number of destinations in S is denoted by $t(S) = |S|$.
The number of cars in S is denoted by $n(S) = \sum_{d=1}^{t(S)} |S_d|$.
Alternatively S can be represented by a $n(S)$-tuple

$$T^S = (d_1, \ldots, d_{n(S)})$$

with $1 \leqslant d_i \leqslant n(S)$. Here d_i denotes the destination of the ith car arriving at the hump yard, i.e. $i \in S_{d_i}$.
The set of all possible problem instances will be denoted by $\mathcal{S}$.

For any destination $S_i \in S$ the numbers of the cars $i \in S_i$ can be seen as the arrival times of the cars. We will assume them to be integers in the range of $1, \ldots, n$. This for example allows us to sort the cars of a destination in linear time, using radix sort or some other linear time sorting scheme for integer valued sets (see for example Knuth [9]).

If the context allows no disambiguation we will simply use n and t for the number of cars or destinations.

While in S we look at the destinations and which cars belong to them, the tuple T^S reflects incoming train and is often a more intuitive way to describe a problem instance. As the two representations for problem instances S and T^S can easily be interchanged, we will use both ways to describe a problem instance, depending which one is more practical in the respective case.

Example 1.2. In the example shown in figure 3 the corresponding problem instance would look as follows:

$$S = \{\{1, 5, 11\}, \{2, 6\}, \{3, 8\}, \{4, 7, 10\}, \{9\}\}$$

with the alternative representation

$$T^S = (1, 2, 3, 4, 1, 2, 4, 3, 5, 4, 1)$$

Now we will give a definition of a valid solution for a TMP instance.

Definition 1.3. A valid solution for $S \in \mathcal{S}$ is given by a tuple

$$L = (L_1, \ldots, L_k)$$

where the

$$L_i = (l_1^i, \ldots, l_{m_i}^i)$$

represent the sorting tracks and we have

$$\forall\, s \in S_d \in S\, \exists i, j : \, l_j^i = s$$

i.e. every car must occur on one sorting track. Furthermore we must have

$$j_1 < j_2 \Rightarrow l_{j_1}^i < l_{j_2}^i \quad \forall i$$

i.e. the car that is placed first on a sorting track must have arrived first in the incoming sequence.

Now let

$$l_h = l_j^i, \textit{ where } i = \min\{g \in \mathbb{N}_0 : \sum_{i'=1}^{g} m_{i'} > h\} + 1$$
$$\textit{and } j = h - \sum_{i'=1}^{i-1} m_i$$

so the l_h represent the sequence we get when concatenating the tracks L_i.

Now for L to be a valid solution we further require

$$l_{i_1}, l_{i_2} \in S_d, \, i_1 < i_2 \Rightarrow l_{i'} \in S_d \quad \forall i_1 < i' < i_2$$

i.e. cars of one destination may not be interrupted by cars of other destinations after concatenating the sorting tracks.

In the definition above we assume the sorting tracks L_i to be ordered in such a way, that they can be pulled out in increasing order. This is mainly for convenience and especially in the case of the online problem (see Chapter 3) we will refrain from this requirement.

Next we will show a different way to represent a solution based on an approach taken by Dahlhaus et al. [5]. The main idea is to look at the ordering of the destinations in the outbound train. For any permutation of the destinations we can define a corresponding solution that results in an outbound train where the destinations appear in the order of the given permutation.

A permutation π of the destinations defines a solution

Algorithm 1 Calculate a solution from a permutation of the destinations

$l \leftarrow 1$
$K \leftarrow 1$
for $i = 1$ **to** t **do**
  Sort $S_{\pi(i)} = (s_1, \ldots, s_m)$ by arrival time
  Choose j minimal with $s_j \geqslant l$
  Add cars $(s_j, \ldots, s_m)$ to L_K
  $l \leftarrow s_m$
  if $j > 1$ **then**
    $K \leftarrow K + 1$
    Add cars $(s_1, \ldots, s_{j-1})$ to L_K
    $l \leftarrow s_{j-1}$
  end if
end for

Definition 1.4. For any permutation π of the numbers $1, \ldots, t(S)$ a solution L_S^π is given by Algorithm 1 such that the destinations of the outbound train appear in the order of π.

What Algorithm 1 actually does is to place the cars of $S_{\pi(1)}$ on sorting track L_1 followed by all cars of $S_{\pi(2)}$ that arrive after the last car of $S_{\pi(1)}$. If all cars of $S_{\pi(2)}$ can be placed this way, we continue with the next destination until there are cars left over that cannot be placed on L_1 as they arrive too early. These are placed on track L_2 where the same procedure is continued until the cars of last destination $S_{\pi(t)}$ are completely assigned.

The algorithm runs in linear time, as we need to consider every destination only once and sorting the integer values of each destination can be done in linear time (using for example Radix sort [9]).

If the order of destinations is fixed, calculating K is easy

Note that Algorithm 1 uses the smallest number of tracks to arrive at a solution where the destinations are ordered in the required way (see [6]).

Another way to look at this was given by [5]. We look at K repetitions of the numbers $1, 2, \ldots, n$. K shall be chosen smallest, such that the repeated series of numbers can still contain the numbers of $S_{\pi(1)}$ followed be the numbers of $S_{\pi(2)}$ and so forth. Here each repetition $1, 2, \ldots, n$ corresponds to one of the sorting tracks. For more information about this representation of a TMP solution the reader shall be referred to [5]. The idea of this solution representation will be illustrated by the following example based on the problem instance from example 1.2.

Example 1.5. For the instance S given in example 1.2 the solution as depicted in figure 3 will look like

$$\pi = (2, 3, 4, 1, 5)$$

When following the steps of Algorithm 1 we first place the cars of destination

$$S_{\pi(1)} = S_2 = \{2, 6\}$$

on L_1, so we have $L_1 = (2, 6)$. The next destination is

$$S_{\pi(2)} = S_3 = \{3, 8\}$$

As car 3 arrives to early, we can only place car 8 on L_1. 3 will be placed on the next track L_2. Next the cars of destination

$$S_{\pi(3)} = S_4 = \{4, 7, 10\}$$

fit completely on L_2 and we continue till we arrive at the final solution

$$L_S^\pi = ((2, 6, 8), (3, 4, 7, 10, 11), (1, 5, 9))$$

If we look at the repetition of the interval $1, 2, \dots, 11$ for three times, we can fit the cars in the following way

$$1\ \underbrace{2\,3\,4\,5\,6}_{S_2}\ 7\ \underbrace{8\,9\,10\,11\,1\,2\,3}_{S_3}\ \underbrace{4\,5\,6\,7\,8\,9\,10}_{S_4}\ \underbrace{11\,1\,2\,3\,4\,5}_{S_1}\ 6\,7\,8\ \underbrace{9}_{S_5}\ 10\,11$$

Observe how the different ways of looking at the solution can easily be interchanged.

Next we will define some more properties of valid solutions as well as optimal solutions:

Definition 1.6. For any solution L let

$K(L)$ is the number of sorting tracks in L

$$K(L) = |L|$$

be the number of sorting tracks used in L. The same way we define for a permutation π

$$K(\pi, S) = K(L_S^\pi)$$

Let $\mathfrak{S}_{t(S)}$ be the symmetric group of all possible permutations of the numbers $1, \dots, t(S)$. Then $\mathfrak{S}_{t(S)}$ represents a set of valid solutions for S.

Now $K(S) = \min_{\pi \in \mathfrak{S}_{t(S)}} K(\pi, S)$ is the minimal number of sorting tracks necessary to sort the train represented by S.

$K(S)$ is the minimal number of sorting tracks necessary to sort S

We call any $\pi^* \in \mathfrak{S}_{t(S)}$ with $K(\pi^*, S) = K(S)$ and the corresponding $L_S^{\pi^*}$ an optimal solution for S.

In the following we want to take a closer look on the structure of a specific solution.

Definition 1.7. Let $S \in \mathbb{S}$, $S = \{S_1, \ldots, S_t\}$ and $L = (L_1, \ldots, L_k\}$ be a solution for S. We say a destination d is split in L if

$$\exists i \neq j, s_1, s_2 \in S_d : s_1 \in L_i \text{ and } s_2 \in L_j$$

i.e. cars from destination d appear on two different sorting tracks in the solution.

$Z(L)$ is the number of split destinations in L

For any solution L we can now define the following property

$$Z(L) = |\{d, \text{ destination } d \text{ is split in } L\}|$$

stating the total number of split destinations.

It is clear, that from the definition of the solution L the cars from a split destination d must appear at the very end of one of the two tracks while the other track has to start with cars from d.

Example 1.8. In the solution L_S^π from example 1.5 the destinations 1 and 3 were split between different sorting tracks. Therefore we have

$$Z(L_S^\pi) = 2$$

$Z(S)$ is the minimal number of split destinations for an optimal solution of S

Definition 1.9. Let $S \in \mathbb{S}$. Then

$$Z(S) = \min_{L \text{ is optimal solution for } S} Z(L)$$

gives the minimal number of split destinations necessary for an optimal solution.

Next we will define sub-instances of a given instance S. This will later be helpful to derive lower bounds on $K(S)$.

1.3.2 *Sub-instances*

Removing cars and destinations leads to sub-instances

Definition 1.10. Let $S \in \mathbb{S}$,

$$S = \{S_1, \ldots, S_n\}$$

We now call a set

$$D = \{S'_{n_1}, \ldots, S'_{n_l}\}$$

with

$$n_i \neq n_j \quad \forall i \neq j$$

and

$$S'_i \subseteq S_i$$

a sub-instance of S.

So a sub-instance is a TMP instance where destinations and cars from the original instance may have been removed. Sub-instances can be treated in the same way as the original instance, but their properties and optimal solutions may vary.

Next we will look at a relation between two sub-instances:

Sub-instances are disjoint if they do not overlap nor share destinations

Definition 1.11. Let $S \in \mathcal{S}$ and

$$D_1 = \{S'_{n_1}, \dots, S'_{n_l}\},\ D_2 = \{S'_{m_1}, \dots, S'_{m_o}\}$$

be sub-instances of S. We call D_1 and D_2 disjoint sub-instances if the following conditions are satisfied:

- They have no destinations in common:

$$n_i \neq m_j \quad \forall i \in \{1, \dots, l\}, j \in \{1, \dots, o\}$$

- The last car of one sub-instance arrives before the first car of the other sub-instance. Therefore we either have

$$x \in S'_{n_i}, y \in S'_{m_j} \Rightarrow x > y \quad \forall i \in \{1, \dots, l\}, j \in \{1, \dots, o\}$$

  or

$$x \in S'_{n_i}, y \in S'_{m_j} \Rightarrow x < y \quad \forall i \in \{1, \dots, l\}, j \in \{1, \dots, o\}$$

The disjoint property will help us to ensure that the two sub-instances will not interfere with one another when solved subsequently.

Example 1.12. As an example for disjoint sub-instances we first take the instance S from example 1.2. Now we consider the sub-instances

$$D_1 = \{\{1, 5\}, \{2, 6\}, \{3\}\}$$

and

$$D_2 = \{\{7, 10\}, \{9\}\}$$

They are disjoint as the last car of D_1 is 6 and therefore arrives before the first car of D_2, which is 7.

Now we want to relate the solutions of the original instance to solutions of its sub-instance.

Definition 1.13. Let $S \in \mathcal{S}$ and D be a sub-instance of S. Furthermore let L be a solution for S. We say

$$L' = (L_{m_1}, \dots, L_{m_{k'}})$$

with $k' \leqslant k$ and $m_i \neq m_j\ \forall i \neq j$ is the sub-solution of L corresponding to D if it contains exactly the cars from the original solution L that also appear in D.

Sub-instances can only have a smaller optimum

Obviously $K(S) \geqslant K(D)$ as the optimal solution cannot use more tracks for a reduced problem instance.

1.3.3 *Interval graphs and overlappings*

In this section we will introduce a relation between problem instances and interval graphs. This relation exists as any destination can be seen as an interval that starts with its first and ends with its last car. This way we can define the corresponding interval graph in the following way:

There is an interval graph that represents the overlapping destinations of S

Definition 1.14. Let $S \in \mathbb{S}$. We now define an interval graph

$$G^S = (S, E)$$

where each node is a destination from S and

$$\exists x, y \in S_i,\ z \in S_j,\ i \neq j,\ x < z < y \Rightarrow \{S_i, S_j\} \in E$$

i.e. there is an edge between the destinations i and j if there is a car of destination j that arrives in between cars of destination i. In this case we say that i and j are overlapping destinations.

Note that while every problem instance has exactly one corresponding interval graph, we can find infinitely many problem instances that correspond to a certain interval graph.

With this relation to interval graphs established, we need some basic definitions from graph theory

Definition 1.15. For $G^S = (S, E)$ we call $C \subseteq S$ a clique iff

$$S_i, S_j \in C \Rightarrow \{S_i, S_j\} \in E \quad \forall i \neq j$$

i.e. all distances in C are overlapping with each other.

We call $C \subseteq S$ a maximal clique iff there is no clique $C' \subseteq S$ with $C \subsetneq C'$, i.e. C cannot be enlarged by adding more destinations.

We call $C \subseteq S$ a maximum clique iff for all possible cliques $C' \subseteq S$ we have $|C| \geqslant |C'|$, i.e. there is no larger clique than C.

Definition 1.16. Let $S \in \mathbb{S}$ and c be any clique in G^S. Then let

$$u(c) = |c|$$

For two cliques c_1 and c_2 let

$$c_1 \cap c_2$$

be the clique that contains only the instances that appear both in c_1 and c_2.

Note that $c_1 \cap c_2$ is also a clique as the subset of a clique must obviously be a clique by its own.

Now we want to relate the properties of the corresponding interval graph back to the original problem.

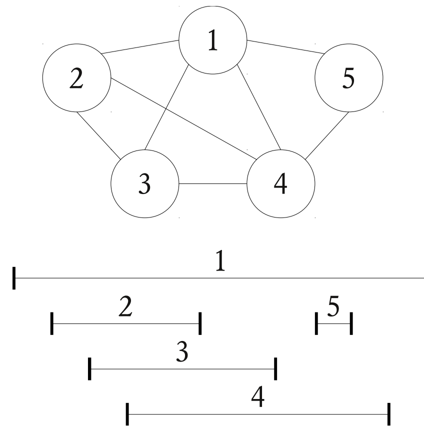


Figure 4: An example for an interval graph with the corresponding intervals

$u(S)$ is the maximum number of mutually overlapping destinations

Definition 1.17. Let $S \in \mathcal{S}$ and let C be a maximum clique of G^S. Then we say

$$u(S) = u(C)$$

is the number of maximal overlapping intervals.

Again this number will help us later to derive bounds on $K(S)$. [8] show that determining maximum cliques in preordered interval graphs can be done in $\mathcal{O}(n)$ time. We can assume that ordering the intervals can be done in linear time as we only deal with integer values for the arrival times (see remarks to Definition 1.1). Furthermore if we know the sequence T^S, we already have the ordering of the instances.

Example 1.18. Again we will illustrate the definitions made above for the problem instance from example 1.2. In figure 4 the corresponding interval graph G^S is depicted as well as the intervals that correspond to the different destinations.

G^S has two maximal cliques

$$C_1 = \{S_1, S_2, S_3, S_4\}$$

and

$$C_2 = \{S_1, S_4, S_5\}$$

where C_1 is a maximum clique. Therefore we have $u(S) = |C_1| = 4$.

1.3.4 *Algorithms*

Definition 1.19. An algorithm alg for TMP is a mapping that returns a valid solution for any problem instance $S \in \mathcal{S}$:

$$alg : S \mapsto L \quad \text{where } L \text{ is valid solution of } S$$

As any algorithm $\mathfrak{alg}$ always produces a solution for a problem instance S, it also yields an upper bound on the optimal number of sorting tracks:

$$K(\mathfrak{alg}(S)) \geqslant K(S)$$

Later in Chapter 3 about online algorithms we will make the further requirement that an algorithm may not possess knowledge about the whole problem instance when assigning a car to a sorting track, but only about the cars up to the current one.

2 OFFLINE PROBLEM

In this chapter we will consider the offline version of TMP, i.e. we will deal with the case that the whole problem instance S is known beforehand. The goal will be to construct a solution that is optimal or at least a close approximation to the optimal solution.

First note the following result regarding the complexity of the general TMP:

Theorem 2.1. *Determining the minimal number of sorting tracks $K(S)$ in general is NP-hard.*

TMP is NP-hard

Proof. See Dahlhaus et al. [5] □

Therefore, in concordance with the general belief, we can say that solving TMP in general is difficult and can not be implemented by any fast (i.e. polynomial time) algorithm.

The next steps will be to derive an Integer Program (IP) Formulation for TMP which will be done in section 2.1. Afterwards we will derive some bounds for $K(S)$ in section 2.2.

2.1 IP FORMULATION

To formulate an TMP instance S as an IP we need to define its objective as well as constraints. In a straight forward approach we will introduce a variable $K \in \mathbb{N}$ that will correspond to the total number of sorting tracks. Obviously this will also be our objective value.

Next we need to think up the necessary constraints. First we can think of a variable x_i for every car $i \in \{1, \ldots, n\}$ that tells us on which of the sorting tracks car i will be placed. So we get the constraints

$$1 \leqslant x_i \leqslant K, x_i \in \mathbb{N} \quad \forall i \in \{1, \ldots, n\}$$

as every car has to be placed on one of the K sorting tracks.

Now we have to encode the constraint that all cars of a certain destination need to appear in a consecutive order. To achieve this we introduce variables y_{d_1,d_2} for all pairs of destinations $d_1, d_2 \in \{1, \ldots, t\}$. Those will be binary variables that are 1, if destination d_1 will appear before d_2 in the outgoing sequence and 0 otherwise. Now we only need to find constraints that link the x_i with the y_{d_1,d_2}.

First note that for every car i the number

$$n \cdot (x_i - 1) + i$$

gives some kind of relative position in the outgoing sequence, i.e. a car i leaves the marshalling yard before car j if and only if

$$n \cdot (x_i - 1) + i < n \cdot (x_j - 1) + j$$

This becomes clear if we consider the following two possible cases. Either the two cars are on different sorting tracks $x_i \neq x_j$. Then obviously the car leaves first, that is on the sorting track that is pulled out earlier (the smaller one). If the cars are on the same sorting track $x_i = x_j$, the car leaves first, that has entered first, i.e. here it only matters if $i < j$.

Now we need to find a linear constraint that lets all cars of one destination either leave before or after all cars of another destination. To achieve this we need an upper bound on the objective value K. As we will see shortly in Theorem 2.2 we can use for example the number of destinations t as an upper bound, as placing each destination on one sorting track always yields a solution. Using this we can construct the desired constraints. We need to consider all possible pairs of destinations

$$d_1, d_2 \in \{1, \dots, t\}, d_1 \neq d_2$$

and all pairs of cars from these destinations

$$i \in S_{d_1}, j \in d_{d_2}$$

then we can formulate for those the following constraints

$$n \cdot (x_i - 1) + i + y_{d_1,d_2} \cdot (n \cdot t) \geqslant n \cdot (x_j - 1) + j$$

as well as

$$n \cdot (x_i - 1) + i \leqslant n \cdot (x_j - 1) + j + (1 - y_{d_1,d_2}) \cdot (n \cdot t)$$

This forces car i to leave before car j if $y_{d_1,d_2} = 1$ and vice versa in the other case.

Putting all this together, we can now formulate any TMP instance S as the following IP:

$$\begin{aligned}
&\min K \\
&\text{s.t. } n \cdot (x_i - 1) + i + y_{d_1,d_2} \cdot (n \cdot t) \geqslant n \cdot (x_j - 1) + j \\
&\qquad\qquad \forall i \in S_{d_1}, j \in S_{d_2}, d_1, d_2 \in \{1, \dots, t\}, d_1 \neq d_2 \\
&\qquad n \cdot (x_i - 1) + i \leqslant n \cdot (x_j - 1) + j + (1 - y_{d_1,d_2}) \cdot (n \cdot t) \\
&\qquad\qquad \forall i \in S_{d_1}, j \in S_{d_2}, d_1, d_2 \in \{1, \dots, t\}, d_1 \neq d_2 \\
&\qquad 1 \leqslant x_i \leqslant K, x_i \in \mathbb{N} \qquad \forall i \in \{1, \dots, n\} \\
&\qquad y_{d_1,d_2} \in \{0, 1\} \qquad \forall d_1, d_2 \in \{1, \dots, t\} \\
&\qquad K \in \mathbb{N}
\end{aligned}$$

Note that this IP formulation is not very efficient to be solved in practice (even when using a closer upper bound than the trivial one) and is therefore more of theoretical interest than of practical value.

2.2 BOUNDS

In this section we will present some bounds on $K(S)$. Using them we can define a range of possible values for our optimum. They will be needed later, for example when analyzing the competitiveness of specific algorithms.

We start this section with the following, trivial upper bound:

Theorem 2.2.

$$K(S) \leqslant t(S)$$

$K(S) \leqslant t(S)$

Proof. As already stated earlier, we can always find a solution by simply assigning each destination its own track. □

The first non trivial bound is based on the total number of cars and was first presented by Dahlhaus et al. [5]:

Theorem 2.3.

$$K(S) \leqslant \lceil \frac{n(S)}{4} + \frac{1}{2} \rceil$$

$K(S) \leqslant \lceil \frac{n(S)}{4} + \frac{1}{2} \rceil$

Proof. For the proof see Dahlhaus et al. [5] □

Later we will see in Section 2.3.1 that this bound is possibly sharp and that it leads to a family of instances for which we can determine solutions easily. But in general the bound often tends to be too large (see the results in Chapter 4).

Theorem 2.4.

$$K(S) \leqslant u(S)$$

$K(S) \leqslant u(S)$

Proof. To show the bound, we need to construct a solution using no more than $u(S)$ tracks. The corresponding interval graph G^S can be colored in linear time using an easy algorithm shown in [13]. With coloring an interval graph we refer to the problem of assigning a color to each vertex such that there is no edge between vertices of the same color.

Using such a coloring we can easily generate a solution by using one sorting track per color. The cars are then placed on the track corresponding to the color that was assigned to it in the coloring process. As two destinations of the same color do not overlap, they can be placed on the same track.

An adapted version of the algorithm in [13] is shown in Algorithm 2.

Now let k be the number of colors Algorithm 2 uses for some instance S. The moment color number k is used, there must be $k-1$ unfinished destinations. These destinations and the one that shall be colored with color k must therefore form a clique. So $u(S) \geqslant k$.

□

This sorting scheme will again be used in the chapter about the online problem in section 3.3.2.

Algorithm 2 Algorithm for optimal destination coloring

Set $isUsed(COLOR)$ to $false$ for all colors
for all Car $c \in T^S$ **do**
  d be the destination of c
  if c is the first car of destination d **then**
    Let $COLOR$ be the first color
    while $isUsed(COLOR)$ **do**
      Let $COLOR$ be the next color
    end while
    Assign $COLOR$ to d
    Set $isUsed(COLOR)$ to $true$
  else if c is the last car of destination d **then**
    Let $COLOR$ be the color assigned to d
    Set $isUsed(COLOR)$ to $false$
  end if
end for

Example 2.5. An example for Algorithm 2 can be seen in Figure 5. It uses the same instance as in Example 1.2. In this case the algorithm does not obtain an optimal solution as we have already seen that there exists a solution that only uses three sorting tracks.

Next we will show some lower bounds for $K(S)$.

Lemma 2.6. *Let* $S \in \mathbb{S}$ *and* L *be a solution for* S. *Then*

$$u(S) - K(L) \leqslant Z(L)$$

Proof. Let D be a sub-instance of S, containing only the destinations that are not split in L and let L' be the sub-solution of L corresponding to D.

Let C be a maximum clique in G^S and let C' be the set remaining after removing all destinations that are split in L. We then have

$$|C| - Z(L) \leqslant |C'|$$

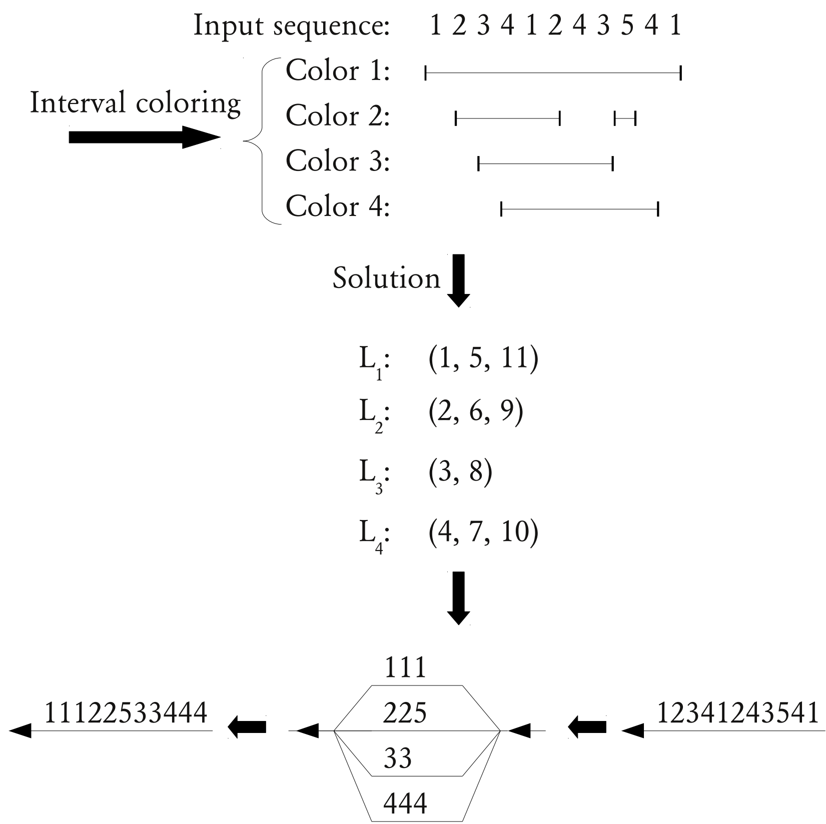


Figure 5: Example for Algorithm 2

as at most $Z(L)$ elements from C might have been removed to obtain C'. As C is a maximum clique in G^S and C' a clique in G^D we get

$$u(S) - Z(L) = |C| - Z(L) \leqslant |C'| \leqslant u(D) \leqslant K(L')$$

where the last inequality holds because there are no split destinations in L' and therefore no overlapping destinations can be placed on the same sorting track. Finally using $K(L') \leqslant K(L)$ (which is obviously true for sub-solutions) we obtain

$$u(S) - Z(L) \leqslant K(L)$$

□

Corollary 2.7. *Let* $S \in \mathbb{S}$*. Then*

$$u(S) - K(S) \leqslant Z(S)$$

Proof. Follows directly from Lemma 2.6 as for any solution L

$$u(S) - K(S) \leqslant u(S) - K(L) \leqslant Z(L)$$

holds and we can choose L to be an optimal solution with minimal splits. □

Lemma 2.8. *Let* $S \in \mathbb{S}$*. Then*

$$K(S) \geqslant Z(S) + 1$$

Proof. Let L be an optimal solution of S with $Z(L) = Z(S)$, i.e. with minimal splits. As any split destination occupies the end of one sorting track and the beginning of the following one, we know that L must use at least $Z(L)$ tracks for all the beginning split destinations plus one additional for the end of the last split destination. Therefore

$$K(S) = K(L) \geqslant Z(L) + 1 = Z(S) + 1$$

□

Using all those inequalities we can finally prove the first lower bound. The idea for this originally comes from [1].

Theorem 2.9.

$$K(S) \geqslant \lceil \frac{u(S) + 1}{2} \rceil$$

$K(S) \geqslant \lceil \frac{u(S)+1}{2} \rceil$

Proof. Combining Corollary 2.7 and Lemma 2.8 we get

$$K(S) \overset{\text{Lem } 2.8}{\geqslant} Z(S) + 1 \overset{\text{Cor } 2.7}{\geqslant} u(S) - K(S) + 1$$

leading to

$$K(S) \geqslant \frac{u(S) + 1}{2}$$

which proves the theorem, as $K(S) \in \mathbb{N}$. □

The bounds from Theorem 2.4 and 2.9 lead to a range of possible values for $K(S)$ that is of size $\lceil \frac{u(S)-1}{2} \rceil$. Note that therefore none of them can be further away from the optimum than by a factor of two.

Next we will derive a new lower bound based on further properties of the problem instance.

Lemma 2.10. *Let* $S \in \mathcal{S}$, D_1 *and* D_2 *be disjoint sub-instances of* S, L *be a solution of* S *and* L_i *be the sub-solution of* L *corresponding to* D_i. *Then*

$$K(L) \geqslant K(L_1) + Z(L_2)$$

Proof. We have to consider 2 cases:

1. CASE: D_1 arrives before D_2

There are $Z(L_2)$ split destinations from D_2 leading to $Z(L_2)$ sorting tracks that must start with a destination from D_2. But those cannot be the sorting tracks used for the cars from D_1 as all of them arrive before the cars from D_2. So in this case the inequality must be fulfilled.

2. CASE: D_2 arrives before D_1

This case follows analogously by considering the $Z(D_2)$ tracks that must end with a destination from D_2 and cannot be the same as any track from L_1.

□

Theorem 2.11. *Let* $S \in \mathcal{S}$, D_1 *and* D_2 *be disjoint sub-instances of* S. *Then*

$$K(S) \geqslant \left\lceil \frac{u(D_1) + u(D_2)}{2} \right\rceil$$

Proof. Let L be any solution for S and L_i be the sub-solution of L corresponding to D_i. Now using Lemma 2.6 and 2.10 we obtain

$$u(D_1) - K(L_1) \overset{\text{Lem. } 2.6}{\leqslant} Z(L_1) \overset{\text{Lem. } 2.10}{\leqslant} K(L) - K(L_2)$$
$$\Rightarrow u(D_1) - K(L) \leqslant K(L_1) - K(L_2)$$

and

$$\begin{aligned} & u(D_2) - K(L_2) \overset{\text{Lem. 2.6}}{\leqslant} Z(L_2) \overset{\text{Lem. 2.10}}{\leqslant} K(L) - K(L_1) \\ \Rightarrow & K(L) - u(D_2) \geqslant K(L_1) - K(L_2) \end{aligned}$$

resulting in

$$\begin{aligned} & u(D_1) - K(L) \leqslant K(L_1) - K(L) \leqslant K(L) - u(D_2) \\ \Rightarrow & u(D_1) + u(D_2) \leqslant 2K(L) \\ \Rightarrow & \lceil \frac{u(D_1) + u(D_2)}{2} \rceil \leqslant K(L) \end{aligned}$$

where the last step holds as $K(L) \in \mathbb{N}$. As this is true for any solution L the theorem is proven.

□

Note that the condition of D_1 and D_2 to be disjoined is crucial:

Example 2.12. We consider the following problem instance

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

with the division into the following, not disjoint sub-instances

$$D_1 = \{\{1, 10\}, \{2, 5\}, \{3, 4\}\}$$

and

$$D_2 = \{\{6, 9\}, \{7, 8\}\}$$

If we would apply Theorem 2.11 a lower bound on the necessary sorting tracks would be

$$K(S) \geqslant \lceil \frac{u(D_1) + u(D_2)}{2} \rceil = \lceil \frac{3 + 2}{2} \rceil = 3$$

but using the permutation

$$\pi = (2, 4, 1, 3, 5)$$

we can achieve a solution using only 2 sorting tracks:

$$K(\pi, S) = 2$$

This can also be seen in figure 6.

Next we will give an example on how Theorem 2.11 can be applied to derive a lower bound on $K(S)$. Furthermore it will illustrate a case where the bound will actually be exact.

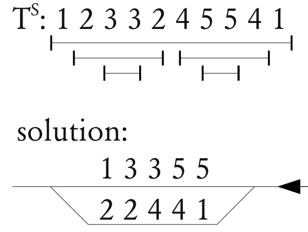


Figure 6: Example for necessity of complete separation of D_1 and D_2

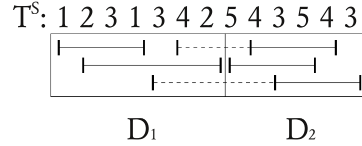


Figure 7: Example for Theorem 2.11

Example 2.13. We consider the problem instance

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

and the disjoint sub-instances

$$D_1 = \{\{1,4\},\{2,7\}\}$$

and

$$D_2 = \{\{8,11\},\{9,12\},\{10,13\}\}$$

(see Figure 7).

By Theorem 2.11 we now get

$$K(S) \geqslant \lceil \frac{u(D_1) + u(D_2)}{2} \rceil = \lceil \frac{2+3}{2} \rceil = 3$$

Furthermore we have $u(S) = 3$ so by Theorem 2.4 we know that

$$3 \leqslant K(S) \leqslant u(S) = 3$$

so S can be sorted optimally using 3 sorting tracks.

Example 2.13 also illustrates a case where the bound from Theorem 2.4 is sharp. On the other hand it gives an instance where the bound from Theorem 2.11 is strictly better than the one derived from Theorem 2.9 as

$$\lceil \frac{u(S)+1}{2} \rceil = \lceil \frac{3+1}{2} \rceil = 2$$

for the specific instance S from Example 2.13.

As the lower bound in Theorem 2.11 depends on the choice of the two sub-instances D_1 and D_2 one can wish to find two sub-instances such that the resulting lower bound is maximized. In the next step we will show that it is possible to find such optimal sub-instances within polynomial time.

First note that the disjoint property of D_1 and D_2 tells us that there must be a certain car c in S that divides D_1 and D_2, i.e. D_2 starts after c and D_1 ends before or at last with c.

Now let D_1^* and D_2^* be disjoint sub-instances of S satisfying

$$\lceil \frac{u(D_1^*) + u(D_2^*)}{2} \rceil = \max_{D_1, D_2 \text{ disjoint sub-instances of } S} \lceil \frac{u(D_1) + u(D_2)}{2} \rceil$$

and let c^* be a car dividing D_1^* and D_2^*.

First assume we already know c^*. In this case we can divide S into two sub-instances D_1' and D_2' where

$$T^{D_1'} = (d_1, \ldots, d_{c^*})$$

and

$$T^{D_2'} = (d_{c^*+1}, \ldots, d_{n(S)})$$

i.e. D_1' consists of the first c^* and D_2' of the last $n(S) - c^* - 1$ cars. Note that D_1' and D_2' do not need to be disjoint, but we know that D_1^* must be a sub-instance of D_1' and D_2^* a sub-instance of D_2'. Our goal will be to determine the quantity

$$u(D_1^*) + u(D_2^*)$$

as we are not interested in the concrete sub-instances D_1^* and D_2^* but only in the resulting lower bound.

For each interval graph $G^{D_1'}$ and $G^{D_2'}$ we can list all maximal cliques in $\mathcal{O}(n)$ time (see [8]). We can now compare all possible pairs (c_1, c_2) with c_1 maximal clique in $G^{D_1'}$ and c_2 maximal clique in $G^{D_2'}$, this way finding a pair that maximizes

$$u(c_1) + u(c_2) - u(c_1 \cap c_2)$$

(the last term resulting from the fact that any destination may only appear in one of the sub-instances).

As the maximum cliques of D_1^* and D_2^* must be subsets of such maximal cliques c_1 and c_2 we will find an optimal solution.

Now we just need to do this for all possible cars c, arriving at the correct guess c^* in at most $n(S)$ steps.

The maximal bound for Thm. 2.11 can be found in polynomial time

Putting all this together we get a total runtime of $\mathcal{O}(n^4)$ as we need to run through $\mathcal{O}(n)$ cars, each time compare $\mathcal{O}(n^2)$ pairs of maximal cliques and calculate the number of shared destinations for each pair of maximal cliques, which can be done in $\mathcal{O}(n)$. Algorithm 3 shows the resulting algorithm.

Algorithm 3 Determine optimal bound for Theorem 2.11

$c \leftarrow 0$
for all car $i \in \{1, \ldots, n-1\}$ **do**
  Divide instance S into D_1', D_2' with $T^{D_1'} = (d_1, \ldots, d_i)$ and $T^{D_2'} = (d_{i+1}, \ldots, d_n)$
  Find all maximal cliques in $G^{D_j'}$ and store them in C_j, for all $j \in \{1, 2\}$
  for all $c_1 \in C_1$ and $c_2 \in C_2$ **do**
    $c \leftarrow \max(c, u(c_1) + u(c_2) - u(c_1 \cap c_2))$
  end for
end for
return c

2.3 EASY INSTANCES

Using the bounds from the last section, we can think of instances where a lower and an upper bound coincide. In this case the decision problem "$K(S) \geqslant k$?" would be easy to answer. In the following families of such instances will be defined. Furthermore we will provide algorithms to efficiently solve the corresponding optimization problem.

2.3.1 *Pair instances with maximal overlapping destinations*

Pair problems have exactly two cars per destination

Definition 2.14. We call a problem instance $S \in \mathbb{S}$ a pair problem instance, iff

$$S_i \in S \Rightarrow |S_i| = 2$$

i.e. there are exactly two cars per destination in S.

A pair problem instance implies that $n(S) = 2 \cdot t(S)$. Now we can define a subset of pair problem instances that is easy to solve:

Corollary 2.15. *Let $S \in \mathbb{S}$ be a pair problem instance and $t(S) = u(S)$ then*

$$K(S) = \lceil \frac{u(S)+1}{2} \rceil$$

Proof. From the assumptions we get that $n(S) = 2t(S) = 2u(S)$. This leads to

$$\lceil \frac{u(S)+1}{2} \rceil \overset{\text{Theorem 2.9}}{\leqslant} K(S) \overset{\text{Theorem 2.3}}{\leqslant} \lceil \frac{n(S)}{4} + \frac{1}{2} \rceil = \lceil \frac{u(S)+1}{2} \rceil$$

□

Corollary 2.15 gives a set of instances for which the bounds in Theorem 2.3 and 2.9 are sharp. Therefore we can easily answer the decision problem. Next we give an algorithm to efficiently

solve the optimization problem, i.e. that computes optimal solutions. It is based on the proof of Theorem 2.3 given in [5]. Algorithm 4 shows the resulting algorithm. It runs in $\mathcal{O}(n)$ time.

Algorithm 4 Algorithm to determine optimal solution for instances from Corollary 2.15

for $i = 1$ to $\lfloor \frac{t(S)}{2} \rfloor$ **do**
  Let $S_{2i-1} = \{x_1, x_2\}$ and $S_{2i} = \{y_1, y_2\}$, $x_1 < x_2$, $y_1 < y_2$
  if $x_2 < y_2$ **then**
    $\pi(2i-1) = 2i-1$
    $\pi(2i) = 2i$
  else
    $\pi(2i-1) = 2i$
    $\pi(2i) = 2i-1$
  end if
end for
if $t(S)$ is odd **then**
  $\pi(t(S)) = t(S)$
end if
return L_S^π

Now it is left to show that for the resulting L_S^π we achieve the optimal number of sorting tracks.

Algorithm 4 solves pair instances with maximal overlappings optimally

Theorem 2.16. *For algorithm 4 we have*

$$K(alg_4(S)) = \lceil \frac{t(S)+1}{2} \rceil$$

Proof. Let $S = \{S_1, \dots, S_{t(S)}\}$ where

$$S_i = \{x_1^i, x_2^i\},\ x_1^i < x_2^i$$

Now define

$$\alpha_i = \max(x_2^{2i-1}, x_2^{2i})$$

and

$$\beta_i = \min(x_2^{2i-1}, x_2^{2i})$$

Next we show via induction that for each solution

$$L = alg_4 = (L_1, \dots, L_k)$$

of algorithm 4 we have

$$L_i = (\dots, \beta_i, \alpha_i)\ \forall 1 \leqslant i \leqslant \lfloor \frac{t(S)}{2} \rfloor$$

That means we always look at the second cars of two destinations and show that the sorting tracks will end with the larger of the two cars, preceded by the smaller one.

To start the induction look at the first track L_1. There are two cases.

CASE 1: $\pi(1) = 1, \pi(2) = 2$

In this case we have $z_1 = 2$. By the ordering of the destinations we will have

$$L_1 = (x_1^1, x_2^1, x_2^2) = (1, \beta_1, \alpha_1)$$
$$L_2 = (x_1^2, \ldots) = (2, \ldots)$$

CASE 2: $\pi(1) = 2, \pi(2) = 1$

This case works the same way as case 1.

Next we assume that the first i sorting tracks look as claimed. Let α_i be of destination d. Then track L_{i+1} has to begin with the first car of destination d as L_i ends with the other one.

Now let d' be the destination of β_{i+1}. By the choice of algorithm 4 this will be the next destination to be placed on L_{i+1}. As $d < d'$ we get

$$x_1^d < x_1^{d'} < x_2^{d'} = \beta_{i+1} < \alpha_{i+1}$$

and therefore the cars can be placed on L_{i+1} as claimed. This finishes the induction.

Now let d be the destination of car $\alpha_{\lfloor \frac{t(S)}{2} \rfloor}$. As we have just seen $L_{\lfloor \frac{t(S)}{2} \rfloor}$ will end with car x_2^d, so we know that $L_{\lfloor \frac{t(S)}{2} \rfloor + 1}$ will begin with car x_1^d. If $t(S)$ is even, then there are no more destinations left to be sorted. Otherwise there is still destination

$$S_{t(S)} = \{x_1^{t(S)}, x_2^{t(S)}\}$$

As

$$x_1^d < x_1^{t(S)} < x_2^{t(S)}$$

we can place both remaining cars on $L_{\lfloor \frac{t(S)}{2} \rfloor + 1}$.

Therefore all cars are sorted on

$$\lfloor \frac{t(S)}{2} \rfloor + 1 = \lceil \frac{t(S) + 1}{2} \rceil$$

tracks.

□

As the proof did not use the fact that $u(S) = t(S)$, algorithm 4 can be used to sort any pair problem instance using $\lceil \frac{t(S)+1}{2} \rceil$ sorting tracks, even if it might not be an optimal solution in case of $u(S) < t(S)$.

2.3.2 *Instances with large disjoint sub-instances*

Next we will look at instances for which the bounds from Theorem 2.4 and Theorem 2.11 coincide. Such an instance was already shown in Example 2.13.

For the two bounds to be equal, there have to be two sub-instances D_1 and D_2 of S such that

$$\lceil \frac{u(D_1) + u(D_2)}{2} \rceil = u(S)$$

which is equivalent to

$$u(D_1) + u(D_2) \geqslant 2 \cdot u(S) - 1$$

So if S has disjoint sub-instances, each having basically the same number of overlapping instances as the original S (one of them may have one less), then S can be sorted optimally using the coloring of Algorithm 2 as shown in Theorem 2.4.

2.3.3 *Fixed number of destinations*

Next we will look at the results we can achieve, if we keep the parameter $t(S)$ bounded by a fixed constant.

Theorem 2.17. *If* $t(S) \leqslant t$ *for a fixed constant* t*, then* $K(S)$ *can be calculated in linear time.*

Proof. For any permutation $\pi \in \mathfrak{S}_{t(S)}$ we can calculate the corresponding $K(L_S^\pi)$ in linear time (see Algorithm 1). There are $t(S)! \leqslant t!$ such possible permutations in total. Therefore checking each of them and choosing the best one can be done in $\mathcal{O}(t! \cdot n(S)) = \mathcal{O}(n(S))$ time. □

2.3.4 *Fixed overlappings*

In a similar fashion to the last section we will now bound the parameter $u(S)$ by a fixed constant and see in this case again, that we can find a polynomial time algorithm to solve the problem

Theorem 2.18. *If* $u(S) \leqslant u$ *for a fixed constant* u*, then* $K(S)$ *can be calculated in polynomial time.*

Proof. For any problem instance S with $u(S) \leqslant u$ we know that

$$u \geqslant u(S) \overset{\text{Theorem 2.4}}{\geqslant} K(S) \overset{\text{Lemma 2.8}}{\geqslant} Z(S) + 1$$

and therefore we know that there is an optimal solution that has at most $u - 1$ split destinations. Each split has to occur between two of the $n(S)$ cars, so there are less than $n(S)^{u-1}$ possibilities how the splits may be distributed.

In [6] it is shown, that one can compute an optimal solution in linear time if one requires the cars of each destination to occur in a fixed order within its destination. Determining the splits beforehand results in fixing the order of cars within each destination. So for each of the $n(S)^{u-1}$ possibilities for fixing the

splits, we can compute the corresponding optimal solution in linear time. Therefore $K(S)$ can be computed in $\mathcal{O}(n(S)^u)$ - i.e. in polynomial - time, as u was chosen to be constant. □

Note that the proof is based on the boundedness of $K(S)$. This also means that we can use the same algorithm if we know that $K(S)$ is bounded in some other way. Notably this means, that the decision problem "$K(S) \leqslant k$?" can be solved in polynomial time for a fixed k.

Fixing t or u leads to polynomial time, but not necessarily fast algorithms

The results from this and the last section are more of theoretical interest than of real practical value as the running times of the obtained procedures can be far from feasible for large choices of t or u, even though the time complexity is polynomial.

3 ONLINE PROBLEM

3.1 INTRODUCTION TO ONLINE COMPUTATION AND COMPETITIVE ANALYSIS

This chapter will be dedicated to the online version of TMP. In online computation we always face a problem where decisions have to be made on several time steps while the full information about the problem is not available beforehand. In the case of TMP we will assume that we have to assign a track to each arriving car while not possessing knowledge about the sequence of the following cars. The only information we can use about the future is whether the current car is the last car of its destination or not. We can imagine that all the last cars carry a flag indicating that no more cars of the same destination will follow. At the end of this section we will show that this assumption is crucial for the model as otherwise we will not be able to construct any reasonable online algorithms for it.

This kind of model is useful if, for example, we really don't know which cars will arrive next or if we want to take into account possible disturbances in the schedule that occur during the marshalling process.

To compare different online algorithms we need some kind of performance measure. In the field of online computation the method of competitive analysis, as for example proposed by Borodin and El-Yaniv [3], has been established as one of the most successful ways to analyze the quality of an algorithm. In competitive analysis we compare the online algorithms results with the optimal offline results, i.e. the best possible solution if all knowledge would have been available in advance.

In our analysis we need to distinguish between deterministic and randomized online algorithms. For deterministic algorithms, the algorithm's reaction to any specific input will always be the same. On the other hand randomized algorithms can utilize some source of randomness, like a random number generator, to generate its output. Therefore the output can vary even while the input stays the same. As there does not have to be a call to the source of randomness, the class of deterministic algorithms is obviously contained in the class of randomized algorithms.

Following the concepts given in [3] we can now relate these ideas to TMP and get the undermentioned definition for the competitiveness of a deterministic online algorithm.

Definition 3.1. A deterministic online algorithm $\mathfrak{alg}$ is c-competitive, if there exists a constant α such that for all problem instances $S \in \mathbb{S}$ we have

$$K(\mathfrak{alg}(S)) \leqslant c \cdot K(S) + \alpha$$

This means a c-competitive deterministic online algorithm for TMP will always get a solution that can be at most worse by a factor of c (and an additive constant α) than the optimal solution.

Often the problem of obtaining the competitive ratio is viewed as a game between the online player and a malicious adversary. The online players goal will be to construct an algorithm that performs well on any input sequence S the adversary will create. The adversary can now use the knowledge of the algorithm to construct a problem instance that maximizes the number of tracks the online algorithm uses, while keeping $K(S)$ small.

Obviously this concept gets more complicated when considering randomized online algorithms. In this case the algorithm's output is not known in advance, making it more difficult to define the actions of the adversary. Borodin and El-Yaniv [3] propose three different adversary models. Of these we will use the concept of the oblivious adversary, who has to construct the input sequence in advance, just possessing knowledge about the possible decisions of the algorithm and their probabilities, but not about the actual decisions the algorithm will take. We can adjust Definition 3.1 accordingly:

Definition 3.2. A randomized online algorithm $\mathfrak{alg}$ is c-competitive, if there exists a constant α such that for all problem instances $S \in \mathbb{S}$ we have

$$\mathbb{E}[K(\mathfrak{alg}(S))] \leqslant c \cdot K(S) + \alpha$$

So for the randomized algorithm it suffices to generate an outcome that can be expected to be no worse than c times the optimum, while the actual output maybe worse.

Having all necessary ideas at hand we can now show that the flag, indicating the last car, we use for our model is absolutely necessary as there could be no competitive online algorithm otherwise.

Marking the last car of each destination is crucial for the online model

Theorem 3.3. *There is no competitive online algorithm for TMP ignoring the knowledge if a car is the last of its destination*

Proof. Assume there exists such a c-competitive, possibly randomized, online algorithm that we will call $\mathfrak{alg}$. Now take the sequence S_1^m with

$$T^{S_1^m} = (1, 2, \dots, m)$$

for an arbitrary $m \geqslant 1$. Obviously the optimal solution for S_1^m will be to place all cars on the same track, leading to $K(S_1^m) = 1$. As alg is c-competitive, there must be a constant α such that

$$\mathbb{E}[K(alg(S_1^m))] \leqslant c + \alpha$$

i.e. there is a positive probability that alg uses at most $c + \alpha$ tracks to sort S_1^m for all $m \geqslant 1$.

Now we take the sequence S_2^m with

$$T^{S_2^m} = (1, 2, \ldots, m, 1, 2, \ldots, m)$$

Having no flag for indicating a last car, the two sequences S_1^m and S_2^m are absolutely identical up to the mth car. Therefore when given S_2^m alg will again place the first m cars on at most $c + \alpha$ tracks with positive probability.

Now choose $m = 2(c + \alpha) + 1$. Using the already obtained results we get a positive probability that alg will place at least 3 cars of different destinations on one of the sorting tracks. But this must lead to an infeasible solution, as a car in the middle of the mentioned track can never connect to its second car.

So any c-competitive online algorithm will produce infeasible solutions with positive probability there can not be such an algorithm. □

In the past competitive analysis has proven to be a very useful tool, as we do not need to take into account factors like the probability distribution over the incoming instances. Furthermore the competitive ratio grants a guaranteed quality of the algorithm's solution (of course this only holds true for deterministic algorithms). Thus the results of this chapter can also be viewed as approximations for the offline problem.

Online algorithms are robust to schedule disturbances

Another interesting property of online algorithms is their inherent robustness. As long as the last car of each destination is still flagged correctly, any disturbance in the incoming sequence of cars will not affect the algorithm in that it will still produce a valid solution. Online algorithms can therefore be called strictly robust, where an algorithm is said to be strictly robust if its solutions do not need to be recovered in any way. Such an approach is also taken for example by [4, 10]. Considering this makes the online problem especially interesting,, as in railway optimization one often deals with unforeseen changes in schedules.

3.2 LOWER BOUND FOR DET. ONLINE ALGORITHMS

In this section we will derive a lower bound for the entire class of deterministic online algorithms for TMP. Using this knowledge we can later construct an optimal deterministic online algorithm, i.e. one that achieves this bound.

As we restrict ourselves to deterministic online algorithms, we can build a bad sequence for any algorithm based on the choices it has made so far.

We will restrict ourselves to pair instances as given in definition 2.14. That means we can show the lower bound, using only instances having two cars per destination.

Head instances are the first cars of a pair instance

Definition 3.4. We call $\widehat{S} \in \mathbb{S}$ a head instance, iff

$$S_i \in \widehat{S} \Rightarrow |S_i| \leqslant 2$$

A pair problem instance $S \in \mathbb{S}$ is called a completion of $\widehat{S}$, iff

$$T^S = (d_1, \dots, d_{n(\widehat{S})}, \dots, d_{n(S)})$$

where

$$T^{\widehat{S}} = (d_1, \dots, d_{n(\widehat{S})})$$

i.e. S begins with the cars of $\widehat{S}$ and where

$$t(S) = t(\widehat{S})$$

i.e. there are no new destinations in S.

Also we define for any deterministic online algorithm $\mathfrak{alg}$:

$$\mathfrak{alg}(\widehat{S}) = \widehat{L}$$

where $\widehat{L}$ is the sub-solution of $L = \mathfrak{alg}(S)$ corresponding to $\widehat{S}$ where S is any completion of $\widehat{S}$.

The head instances will help to construct bad instances for online algorithms car by car. Note that $\widehat{L}$ is not dependent on the choice of the completion of $\widehat{S}$ as we only deal with online algorithms.

Definition 3.5. Let $L = \mathfrak{alg}(S)$ be a solution to $S \in \mathbb{S}$. We call a track $L_i \in L$ closed, iff

$$L_i = (x_1, \dots, x_m),\ x_m \in S_j \in S$$

and

$$\exists L_{i'} = (y_1, \dots, y_{m'}) \in L,\ y_1 \in S_j,\ i' \neq i$$

i.e. L_i is closed if there exists another sorting track that begins with cars from the same destination as L_i ends.

Lemma 3.6. *Let $\widehat{S} \in \mathbb{S}$ and $\widehat{S'} \in \mathbb{S}$ be two head instances such that $\widehat{S}$ is a sub-instance of $\widehat{S'}$. Then we have*

$$L_i \in \widehat{L} = \mathfrak{alg}(\widehat{S}) \text{ is closed track}$$

implies

$$L_i \in \widehat{L'} = \mathfrak{alg}(\widehat{S'}) \text{is closed track}$$

i.e. a closed track remains closed and there will be no further cars placed upon it by $\mathfrak{alg}$.

Proof. Let

$$L_i = (x_1, \dots, x_m) \in \widehat{L}$$

be a closed track in $\widehat{L}$, i.e. there is another track

$$L_{i'} = (y_1, \dots, y_{m'})$$

with $x_m, y_1 \in S_j \in \widehat{S}$. Now assume $\mathtt{alg}$ adds some of the new cars of $\widehat{S}'$ to L_i i.e. there is a track

$$L_i' = (x_1, \dots, x_m, \dots, x_{m''}) \in \widehat{L}'$$

instead of L_i. But then the car x_m could never connect to the car y_1 of its destination so the solution will not be valid. So track L_i must remain the same and will obviously still be a closed track. □

Definition 3.7. For any problem solution L let

$$C(L) = |\{L_i \in L, L_i \text{ is closed}\}|$$

be the number of closed tracks in L and

$$D(L) = |\{L_i \in L, |L_i| = 2\}|$$

be the number of tracks in L that accommodate exactly 2 cars.

In the following we will construct a bad head instance for an arbitrary deterministic online algorithm $\mathtt{alg}$. At each step of this construction we will make sure that for the head sequence $\widehat{S}$ and its solutions $\widehat{L} = \mathtt{alg}(\widehat{S})$ will satisfy

$$\begin{aligned} &1.\ |L_i| \leqslant 2\ \forall L_i \in \widehat{L} \\ &2.\ L_i = (x, y) \text{ or } L_i = (y),\ y \in S_j \in \widehat{S},\ |S_j| = 2 \\ &\quad \Rightarrow L_i \text{ is closed} \end{aligned} \tag{3.1}$$

i.e. there will be at most 2 cars per track and any track that ends with a car of a destination where both cars are sent in $\widehat{S}$ will be closed. Note the following implication

Lemma 3.8. *Condition 3.1 implies*

$$L_i = (x, y) : x \in S_{j_1} \in \widehat{S},\ y \in S_{j_2} \in \widehat{S} \Rightarrow j_1 \neq j_2$$

i.e. there are no two cars of the same destination on any sorting track in L.

Proof. Assume there would be such a sorting track L_i, i.e.

$$L_i = (x, y) : x, y \in S_j \in \widehat{S}$$

Then by point 2 in condition 3.1 we get that L_i must be closed. But for L_i to be closed there must be a track $L_{i'}$, $i' \neq i$ beginning with a car $z \in S_j$. But this means $|S_j| \geqslant 3$ which cannot be as we have restricted ourselves to pair problem instances. □

Furthermore we will make sure that in any step of our construction either one of the two following conditions will hold:

$$D(\widehat{L}) = C(\widehat{L}) \tag{3.2}$$

or

$$\begin{aligned} &1.\ \exists L_i \in \widehat{L},\ L_i = (x, y),\ x \in S_j \in \widehat{S},\ |S_j| = 1 \\ &2.\ D(\widehat{L}) = C(\widehat{L}) + 1 \end{aligned} \tag{3.3}$$

Point 1 in condition 3.3 states that there is a sorting track in $\widehat{L}$ that carries 2 cars where the first one is of a destination whose second car is not sent in $\widehat{S}$.

Lemma 3.9. *Let* $\mathfrak{alg}$ *be any deterministic online algorithm and let* $\widehat{S} \in \mathcal{S}$ *be a head instance satisfying condition 3.1 as well as condition 3.3. Then there exists a head instance* $\widehat{S}'$ *also satisfying condition 3.1 and either condition 3.2 or condition 3.3. Furthermore* $\widehat{S}'$ *will satisfy*

$$C(\mathfrak{alg}(\widehat{S}')) = C(\mathfrak{alg}(\widehat{S})) + 1$$

Proof. Let $L_i = (x, y)$ be the track given by point 2 in condition 3.3, with $x \in S_j \in \widehat{S}$. Now set

$$\widehat{S}' = (\widehat{S} \setminus \{S_j\}) \cup \{S_j \cup \{n(\widehat{S}) + 1\}\}$$

i.e. we add the second car of destination S_j to the end of $\widehat{S}$.

Let

$$L_{i'} \in \widehat{L}' = \mathfrak{alg}(\widehat{S}')$$

be the sorting track upon which $\mathfrak{alg}$ places the new car $n(\widehat{S}')$ (all other sorting tracks will remain the same as $\mathfrak{alg}$ is an online algorithm). Assume $i' = i$. In this case

$$L_i = (x, y, n(\widehat{S}'))$$

where $x, n(\widehat{S}') \in S_j$ but $y \notin S_j$ (see Lemma 3.8). In this case $\widehat{L}'$ would be no valid solution, so $i' \neq i$.

But now by definition 3.5 $L_{i'}$ must be closed. So we get

$$C(\widehat{L}') = C(\widehat{L})) + 1$$

Using the same argument as before we can also assure that $|L_{i'}| \leqslant 2$ as otherwise by Lemma 3.8 we know that $L_{i'}$ will carry three cars of three different destinations. But the middle car will never be able to connect to its second car making the solution invalid.

This way we have also verified condition 3.1 (note that the only changes may have appeared in track $L_{i'}$). Now we can have the two following cases depending on the question if $L_{i'}$ now carries one or two cars:

CASE 1: $D(\widehat{L'}) = D(\widehat{L})$
In this case we have

$$D(\widehat{L'}) = D(\widehat{L}) = C(\widehat{L}) + 1 = C(\widehat{L'})$$

so condition 3.2 is fulfilled and we are done.

CASE 2: $D(\widehat{L'}) = D(\widehat{L}) + 1$
This implies

$$D(\widehat{L'}) = D(\widehat{L}) + 1 = C(\widehat{L}) + 2 = C(\widehat{L'}) + 1$$

so only point 1 in condition 3.3 remains to be verified.

Assume $L_{i'}$ does not satisfy point 1 in condition 3.3. Now assume $|L_{i'}| = 1$ which would mean that $D(\widehat{L'}) = D(\widehat{L})$ so we know that $L_{i'}$ must be of the form

$$L_{i'} = (x, n(\widehat{S'}))\,,\ x \in S_{j'} \in S,\ |S_{j'}| = 2$$

but then $L_{i'}$ must have been a closed track in $\widehat{L}$ by point 2 of condition 3.1. Therefore $\mathtt{alg}$ could not have placed car $n(\widehat{S'})$ on $L_{i'}$ by Lemma 3.6.

So condition 3.3 must be satisfied which completes the proof. □

Lemma 3.10. *For any* $t \in \mathbb{N}$ *and any deterministic online algorithm* $\mathtt{alg}$ *there exists a head instance* $\widehat{S} \in \mathbb{S}$ *with*

$$t(S) = t = K(\mathtt{alg}(\widehat{S}))$$

Proof. We will inductively generate such a sequence, depending on the choices of $\mathtt{alg}$. We will need to keep track of several of the aforementioned conditions, so our induction hypothesis will look as follows:

INDUCTION HYPOTHESIS For some $t \in \mathbb{N}$ there exists a head instance $\widehat{S^t} \in \mathbb{S}$ that satisfies condition 3.1 as well as condition 3.2.

INDUCTION BASIS We begin with the case $t = 1$. Obviously our head instance will be

$$\widehat{S^1} = \{\{1\}\}$$

for which condition 3.1 and 3.2 will be trivially satisfied as there will only be one sorting track with one car in $\widehat{L^1} = \mathtt{alg}(\widehat{S^1})$ and no closed tracks. Therefore the induction hypothesis is fulfilled for the case $t = 1$.

INDUCTION STEP Now assume the induction hypothesis is fulfilled for $t \in \mathbb{N}$ and let $\widehat{S^t}$ be a head instance satisfying the conditions 3.1 and 3.2.

Now we enlarge $\widehat{S^t}$ by a new destination whose first car is added to the end of $\widehat{S^t}$:

$$\widehat{S^{t+1,1}} = \widehat{S^t} \cup \{\{n(\widehat{S^t}) + 1\}\}$$

As only adding a new destination might not be enough to ensure the induction hypothesis, we will later generate a whole series of $\widehat{S^{t+1,i}}$ by adding cars of the already used $t+1$ destinations. This will be done until the series reaches a point where the induction hypothesis is fulfilled (which will always happen as we will show later). This element of the series will then be chosen as the new $\widehat{S^{t+1}}$. Note that this is the only point in the construction where a new destination is added, so $t(S^t) = t$ will always be satisfied.

Let $L_i \in \widehat{L^{t+1,1}} = alg(\widehat{S^{t+1,1}})$ be the track upon which alg places the new car $\widehat{S^{t+1,1}}$. As L_i ends with $\widehat{S^{t+1,1}} \in S_{t+1} \in S^{t+1,1}$ and $|S_{t+1}| = 1$ point 2 of condition 3.1 is fulfilled (remember that all other tracks than L_i remain unchanged).

Assuming that $|L_i| = 3$ would mean that L_i will look like

$$L_i = (x, y, n(\widehat{S^{t+1,1}}))$$

with cars x, y being of different destinations by Lemma 3.8 and car $n(\widehat{S^{t+1,1}})$ being a completely new destination. But then $\widehat{L^{t+1,1}}$ cannot be a valid solution as y could never connect to its second car.

So condition 3.1 must be fulfilled for $\widehat{S^{t+1,1}}$.

As the car $n(\widehat{S^{t+1,1}})$ is of a new destination and is the only car of this destination in $\widehat{S^{t+1,1}}$, there can be no new closed destinations in $\widehat{L^{t+1,1}}$:

$$C(\widehat{L^{t+1,1}}) = C(\widehat{L^t})$$

Now we must look at the following two cases:

CASE 1: $D(\widehat{L^{t+1,1}}) = D(\widehat{L^t})$

This implies that

$$D(\widehat{L^{t+1,1}}) = D(\widehat{L^t}) = C(\widehat{L^T}) = C(\widehat{L^{t+1,1}})$$

so condition 3.2 is fulfilled and we are done by choosing

$$\widehat{S^{t+1}} = \widehat{S^{t+1,1}}$$

CASE 2: $D(\widehat{L^{t+1,1}}) = D(\widehat{L^t}) + 1$

In this case we get that

$$D(\widehat{L^{t+1,1}}) = D(\widehat{L^t}) + 1 = C(\widehat{L^t}) + 1$$

so point 2 of condition 3.3 is satisfied.

Furthermore we know that

$$L_i = (x, n(\widehat{S^{t+1,1}}))$$

i.e. L_i must be a track with two destinations or otherwise we would be in Case 1. Now assume that

$$x \in S_j \in \widehat{S^{t+1,1}}, \ |S_j| = 2$$

then for x to be able to connect to the second car of its destination there must be another track $L_{i'}$ ending with a car of destination S_j. But this implies that L_i was a closed track in $\widehat{L^t}$ and therefore the car $n(\widehat{S^{t+1,1}})$ could not have been placed on L_i. So we must have $|S_j| = 1$ and condition 3.3 is satisfied.

Now we can use Lemma 3.9 to generate a sequence of head instances in the following way:

$$\widehat{S^{t+1,i+1}} \text{ is the instance generated by Lemma 3.9,}$$
$$\text{if } \widehat{S^{t+1,i}} \text{ satisfies condition 3.3}$$

If the sequence $(\widehat{S^{t+1,i}})$ has a last element $\widehat{S^{t+1,i^*}}$, then this instance must satisfy condition 3.2 by Lemma 3.9. So in this case we are done by choosing

$$\widehat{S^{t+1}} = \widehat{S^{t+1,i^*}}$$

Now assume that there is no such last element. But by Lemma 3.9 we have

$$C(alg(\widehat{S^{t+1,i+1}})) = C(alg(\widehat{S^{t+1,i}})) + 1$$

which is impossible as there can be at most $t+1$ closed tracks, i.e. one for each destination. This finishes the induction.

From the induction we know there exists a head instance $\widehat{S^t}$ for all $t \in \mathbb{N}$ that satisfies

$$D(alg(\widehat{S^t})) = C(alg(\widehat{S^t}))$$

As for each closed track there need to be 2 cars of the same destination, so there must be $C(alg(\widehat{S^t}))$ destinations with two cars and $t(\widehat{S^t}) - C(alg(\widehat{S^t}))$ destinations with one car in $\widehat{S^t}$.

This results in a total number of cars of

$$n(\widehat{S}^t) = 2 \cdot C(\mathfrak{alg}(\widehat{S}^t)) + t(\widehat{S}^t) - C(\mathfrak{alg}(\widehat{S}^t))$$

We also know there are $D(\mathfrak{alg}(\widehat{S}^t))$ tracks that carry two cars which leaves

$$n(\widehat{S}^t) - 2 \cdot D(\mathfrak{alg}(\widehat{S}^t)) = t(\widehat{S}^t) - C(\mathfrak{alg}(\widehat{S}^t))$$

tracks that carry only one car (no track can carry more than two cars by condition 3.1). So in total there must be

$$K(\mathfrak{alg}(\widehat{S}^t)) = D(\mathfrak{alg}(\widehat{S}^t)) + t(\widehat{S}^t) - C(\mathfrak{alg}(\widehat{S}^t)) = t(\widehat{S}^t)$$

tracks.

□

In Figure 8 one can see how the first four cars would be chosen in the sequence generated in Lemma 3.10 depending on the choices of the deterministic algorithm.

Theorem 3.11. *There is no deterministic online algorithm for TMP that is better than 2-competitive.*

No det. online algorithm is better than 2-competitive

Proof. We show that for each number $k \in \mathbb{N}$ there is a sequence of cars such that any online algorithm $\mathfrak{alg}$ will be at least $\frac{2k-1}{k}$ competitive.

We choose $t = 2k - 1$. Now by Lemma 3.10 we know there is a head instance $\widehat{S}^t$ with

$$t(\widehat{S}^t) = t = K(\mathfrak{alg}(\widehat{S}^t))$$

Now let S^t be any completion of $\widehat{S}^t$ which means that

$$t(S^t) = t = K(\mathfrak{alg}(\widehat{S}^t)) \leqslant K(\mathfrak{alg}(S^t))$$

as $\widehat{S}^t$ is a sub-instance of S^t.

Using Theorem 2.3 we know that

$$K(S^t) \leqslant \left\lceil \frac{n(S^t)}{4} + \frac{1}{2} \right\rceil = \left\lceil \frac{4k-2}{4} + \frac{1}{2} \right\rceil = k$$

Therefore $\mathfrak{alg}$ can be at most $\frac{2k-1}{k}$ competitive. As we can choose k arbitrarily large, we get the following lower bound for the competitive ratio of $\mathfrak{alg}$:

$$\lim_{k \to \infty} \frac{2k-1}{k} = 2$$

□

In the next step we want to analyze the quality of different algorithms for the online problem

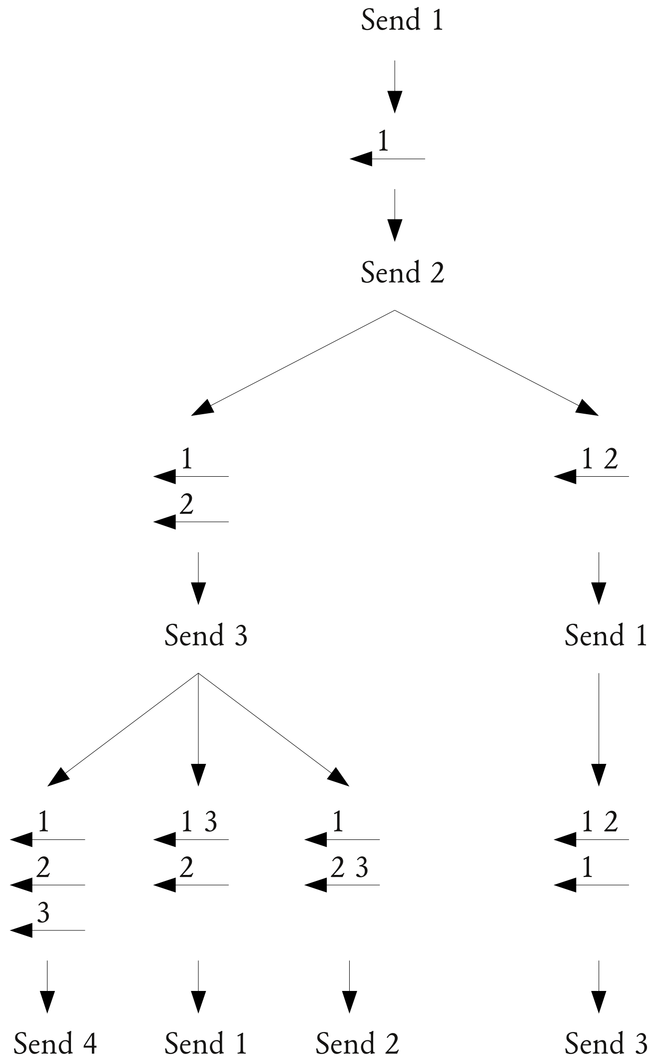


Figure 8: Example for the first four cars of the malicious sequence depending on the choices of the algorithm

3.3 ALGORITHMS

3.3.1 *Split*

A main decision that has to be made by an algorithm for TMP is when to split a destination between two sorting tracks. We will now briefly discuss an online algorithm that always decides to split a destination as early as possible and will see that this behavior is not competitive.

The Split algorithm ([1]) will work by placing each car on the earliest sorting track possible (if there is not already one track that ends with the same destination - in which case the car would be placed there). Note that there are the following three possibilities where a car of destination d cannot be placed on track i:

- Track i ends with a destination d' different to d that is already split. Then placing the new car on i would make it impossible to join destination d'.
- Track i has already two different destinations d_1 and d_2 placed upon it, where $d_2 \neq d$ is the destination of the last car on i. Furthermore we assume that the last car of d_2 was not sent yet. Then placing the new car on i would make it impossible to join destination d_2.
- Placing the new car on i would create a cycle. This means there is a sequence of sorting tracks $(L_{n_j})_{1 \leqslant j \leqslant J}$, where $n_J = i$ that needs to be joined consecutively. If now the first car of L_{n_1} is of destination d the sorting tracks would need to be joined in a circle for all destinations to be joined. Also see Example 3.12 for an illustration of this case.

Example 3.12. Consider the problem instance S with

$$T^S = (1, 2, 3, 1, 2, 3)$$

Here Split would place the first two cars 1 and 2 on sorting track L_1, then the cars 3 and 1 on track L_2. The next car 2 would again be placed on L_1. Until now, we need to join the tracks in the order (L_2, L_1) for the cars of destination 1 to connect in the outgoing sequence. Therefore we cannot place the last car 3 on L_1 as this would also require us to join the tracks in the order (L_1, L_2) for destination 3 to be joined. To avoid this cycle, the last car must be placed on L_3.

Algorithm 5 now shows how the Split algorithm works. Note that we do not bother with the details on how to check if a car can be placed upon a certain track i as these details will not be of interest in the rest of this section.

We now show that Split is uncompetitive

Algorithm 5 The Split algorithm

for all arriving car d **do**
  if ∃ sorting track i ending with cars of same destination as d **then**
    Place d on i
  else
    for all already used sorting track i **do**
      if d can be placed on i **then**
        Place d on i
      end if
    end for
    if d was not placed so far **then**
      Place d on new sorting track
    end if
  end if
end for

Split is not competitive

Theorem 3.13. *There does not exist a constant* c *such that Split is* c*-competitive*

Proof. Assume there is such a constant c. Now choose $t = 3 \cdot c + 1$ and consider the following problem instance

$$S = \{\{1,4\}, \{2, 2 \cdot t\}, S_i \mid 3 \leqslant i \leqslant t\}$$

where for $3 \leqslant i < t$ we have

$$S_i = \{2 \cdot i - 3, 2 \cdot i\}$$

and for $i = t$

$$S_t = \{2 \cdot t - 3, 2 \cdot t - 1\}$$

i.e. the incoming sequence will look as follows

$$T^S = (1,2,3,1,4,3,5,4,6,5,\dots,t,t-1,t,2\}$$

Given this sequence, Split will place all the cars on the first available sorting track. So the first two cars 1 and 2 will be placed on L_1. No more cars can be placed here, as the second car of destination 2 arrives last. The next two cars 3 and 1 will be placed on L_2. Again no more cars can be placed here as L_2 now has to connect to L_1 so that destination 1 can be joined. The same way all other cars are placed in tuples on the sorting tracks. The second car of destination t will therefore be placed on track L_t.

Also see Figure 9 for an illustration on how Split will distribute instance S to the sorting tracks.

So Split uses in total

$$K(\text{Split}(S)) = t = 3 \cdot c + 1$$

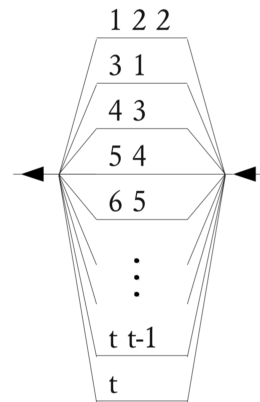


Figure 9: Malicious problem instance for the Split algorithm

sorting tracks. Note that $u(S) = 3$, therefore by Theorem 2.4 the optimal solution needs at most 3 sorting tracks. This leads to a lower bound on the competitive ratio for Split of

$$c \geqslant \frac{K(\text{Split}(S))}{K(S)} = \frac{3 \cdot c + 1}{3} > c$$

which is a contradiction. □

It seems as if the decision to split a destination in the online case always opens a way to construct a malicious sequence that exploits the fact that any split destination leads to a track that is unusable in the future sorting process. Therefore we will now consider an algorithm that never splits any destination.

3.3.2 *Unsplit*

As we have already seen in the proof of Theorem 2.4, we can sort any instance S to $u(S)$ sorting tracks. Note that Algorithm 2 decides about a destinations coloring the moment its first car arrives, without looking at the rest of the train. The only additional knowledge Algorithm 2 needs, is the information if a car is the last of its destination, which is available in our online scenario (see Theorem 3.3). Therefore we can use this algorithm to make an online decision about the placement of the cars. Algorithm 6 shows a version of Algorithm 2 that has been modified for the online scenario. The algorithm is named Unsplit, as it never divides a destination between two sorting tracks in contrast to the Split algorithm in Section 3.3.1.

Algorithm 6 The Unsplit algorithm

```
for all arriving car d do
  if ∃ track x with cars of same destination as d then
    z ← x
  else
    if ∃ track x marked as open then
      z ← x
    else
      open a new track z
    end if
  end if
  place d on z
  if ISLAST(d) then
    mark z as open
  else
    mark z as closed
  end if
end for
```

Theorem 3.14. *Unsplit is 2-competitive.*

Proof. We have already seen in Theorem 2.4 that Unsplit uses at most $K(S)$ sorting tracks. Using this and Theorem 2.9 we get

$$\frac{K(\text{Unsplit}(S))}{K(S)} \leqslant \frac{u(S)}{\lceil \frac{u(S)+1}{2} \rceil} \leqslant 2$$

for any problem instance $S \in \mathcal{S}$. Unsplit is therefore 2-competitve. □

Unsplit is 2-competitive and therefore optimal

As by Theorem 3.11 there cannot be an algorithm that achieves a better competitve ratio, we know that Unsplit is optimal in terms of competitive analysis.

3.4 RANDOMIZED ONLINE ALGORITHMS

Now that we have found an optimal algorithm from the class of deterministic online algorithms, it would be interesting to know if we can derive similar results for randomized online algorithms. Unluckily showing lower bounds for randomized online algorithms becomes much harder, as we do not know the algorithms choices in advance. Therefore constructing a malicious sequence as before is not possible, at least not in the deterministic way we used in Section 3.2.

For example we can find a lower bound for randomized online algorithms in the following way:

Theorem 3.15. *There is no randomized online algorithm for TMP that is better than $\frac{5}{4}$-competitive.*

Proof. We will use Yao's principle (see [3]), i.e. we will generate a probability distribution over different problem instances and show that the expected outcome over these instances will not be better than $\frac{5}{4}$ for any deterministic online algorithm.

The input sequence will be as follows: with probability $\frac{1}{2}$ we will send either the problem instance S_1 with

$$T^{S_1} = (1,2,3,1,2,3)$$

or S_2 with

$$T^{S_2} = (1,2,3,2,1,3)$$

Note that by Theorem 2.9 no algorithm can sort these instances with less than 2 sorting tracks as $u(S_1) = u(S_2) = 3$.

Next we show that any deterministic online algorithm can only sort either one of the two instances to 2 sorting tracks. After the arrival of the first three cars, the algorithm could have made three different choices on how to place the cars without exceeding 2 sorting tracks. Either place car 1 and 2 on the same track or 1 and 3 or 2 and 3. Depending on the choice made here the algorithm will have to open a new track for at least one of the sequences S_1 or S_2. Figure 10 shows the possible options for placing the remaining cars for each problem instance.

So any deterministic algorithm can only achieve 2 sorting tracks with a probability of $\frac{1}{2}$ and therefore the expectation of its number of sorting tracks is at least

$$\frac{1}{2} \cdot 2 + \frac{1}{2} \cdot 3 = \frac{5}{2}$$

As we know that both instances can optimally be sorted using 2 tracks, this leads to an expected competitive ratio of

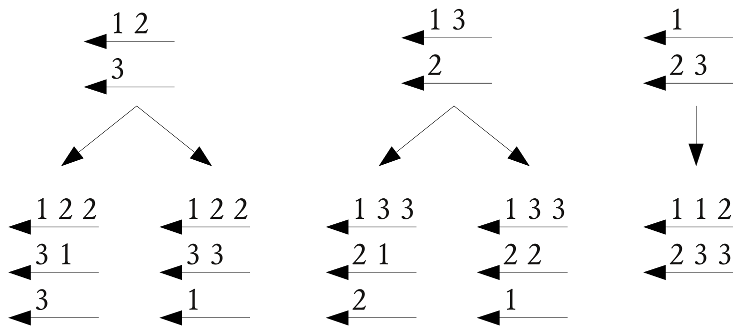
$$\frac{\frac{5}{2}}{2} = \frac{5}{4}$$

which is a lower bound for the competitive ratio of any randomized online algorithm for TMP by Yao's principle.

□

Obviously a lower bound of $\frac{5}{4}$ is a rather weak result compared to the bound of 2 we get by Theorem 3.11. By using more and longer problem instances we might get better results. But then we still need to find a randomized algorithm that achieves a better competitive ratio than 2. This task is difficult as any choice to split a destination opens a way to construct a malicious sequence for this algorithm. So this question remains open for future research.

Sequence S_1: (1,2,3,1,2,3)



Sequence S_2: (1,2,3,2,1,3)

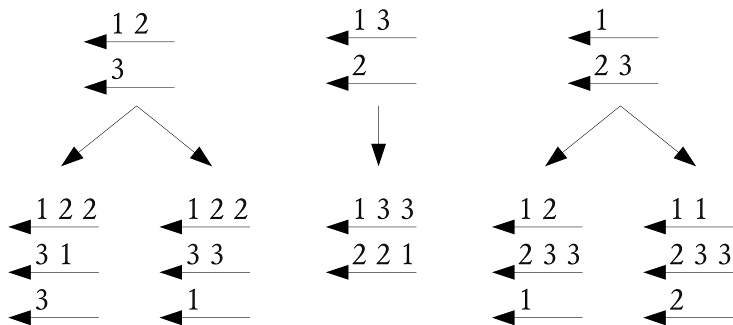


Figure 10: Possible choices for S_1 and S_2

4 COMPUTATIONAL RESULTS

4.1 UNDERLYING PROBABILITY SPACE

To perform any kind of average case analysis we first need to define a probability space of instances so that we can calculate expectations on the performance of our algorithms. As we want to draw random instances from this space for experimental use, we further need to construct an algorithm that will produce random instances according to the specified distribution.

Obviously an uniform distribution on all instances will not work as the size of the instances is unbounded. This leads us to defining instance spaces with a fixed instance size:

Definition 4.1. Let $\mathcal{S}^n$ be the space of all TMP instances with exactly n cars. That means $S \in \mathcal{S}^n \Leftrightarrow n(S) = n$.

This now leads us to the definition of a probability space based on $\mathcal{S}^n$

Definition 4.2. In the following we will use the probability space $\mathscr{S}^n = (\Omega, \mathcal{F}, P)$ with

$$\begin{aligned} \Omega &= \mathcal{S}^n \\ \mathcal{F} &= 2^{\mathcal{S}^n} \\ P(A) &= \frac{\#A}{\#\Omega} \end{aligned}$$

So we will consider the discrete uniform distribution on $\mathcal{S}^n$.

We will consider random instances drawn uniformly form all instances with length n

Now we need to construct a method to choose uniformly from $\mathcal{S}^n$. To do so, we will take a look at the number of possible instances that are left when fixing a certain number of cars in the sequence. This way we will get the probabilities for each possible choice for each car of the sequence. Therefore the algorithm can choose the cars one by one according to the obtained probabilities.

For example in the case of $\mathcal{S}^3$ we have 5 possibilities in total. If we choose destination 2 as the second car we are left with 3 possibilities, if choosing destination 1 there are 2 possibilities left (see figure 11). So in the case of $n = 2$ we would first set car number 1 to be of destination 1 (this is always the case) and then choose destination 1 with probability $\frac{2}{5}$ and destination 2 with probability $\frac{3}{5}$.

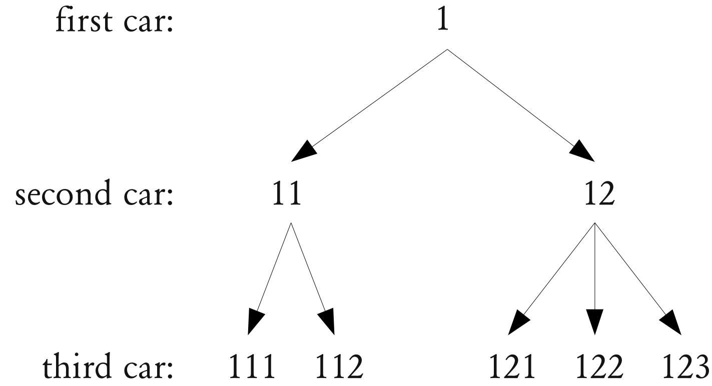


Figure 11: Possibilities when generating a sequence in $\mathcal{S}^3$

So after fixing the first l cars, we want to calculate the number of possibilities for the remaining $n - l$ cars. In the following we will call this final part of a train a tail-sequence.

For each car in the sequence we can either choose one of the destinations that were already sent (first case) or the next new destination (second case). If we decide for the first case, it does not matter, which of the destinations is actually sent as will be shown now:

Lemma 4.3. *If we have already fixed the first $l-1$ cars and have used t different destinations so far ($t \leqslant l-1$), then the number of possible tail-sequences stays the same for any choice of the lth car from $\{1, \ldots, t\}$.*

Proof. A tail-sequence that is a valid if we choose the lth car to be of destination $d \in \{1, \ldots, t\}$ remains valid for any other choice $d' \in \{1, \ldots, t\}$. So the total number of possible tail-sequences is always the same. □

Each of the old destinations can appear with the same probability

This means the first thing we need to decide is whether we choose a car of a new destination or of one of the destinations that were already used. In the later case we can choose the destination uniformly from $\{1, \ldots, t\}$.

So at each step it only matters if we add a new destination or if the number of used destinations t stays the same. Therefore it should be sufficient to keep track of the number of already sent cars, which we denote by l, and the number t of destinations that we used so far. We now define a function P that works on exactly these parameters and that will later be shown to produce the number of possible tail-sequences.

Definition 4.4. For $t \leqslant l$ let

$$P(t, l) = P(t+1, l+1) + t \cdot P(t, l+1)$$

where

$$P(t, l) = 1 \quad \forall l \geqslant n$$

Listing 1: Matlab code to generate a table with values for the possibility function P

```
for l=n:-1:1
    for t=1:l
        if (l>=n)
            P(t,l)=1;
        else
            P(t,l)=P(t+1,l+1)+t*P(t,l+1);
        end
    end
end
```

P is well defined, as the parameter l is growing with each recursion, so it must be equal to n after a finite time.

Theorem 4.5. $P(t, l)$ *gives us the number of possible tail-sequences if we fix the first* l *cars and have used* t *destinations so far.*

Proof. If $l = n$ we have fixed all n cars, so our only choice for the tail-sequence is the empty sequence. Therefore P is correct in this case by definition.

We now want to show that P gives the correct results by recursion. Therefore assume that $P(t, l)$ is correct for all $l \geqslant L + 1$ and all $t \leqslant l$, for some $L < n$.

Then for any $t \leqslant L$ we want $P(t, L)$ to be the number of possible tail-sequences if we used t destinations so far and fixed the first L cars. As we have seen before we can either choose the $L + 1$-st car to be of destination $t + 1$ or to be of one of the destinations in $\{1, \dots, t\}$. All choices from the later case are equal by Lemma 4.3. As we assumed P to yield correct results when fixing the first $L + 1$ cars, we now have in total

$$P(t + 1, L + 1) + t \cdot P(t, L + 1)$$

possible tail-sequences.

Now the claim follows by recursion. □

As calculating P by recursion would take up to 2^n steps we need a more clever way to get the values of P. As there are less than n^2 possible values for P we can generate a table of all values in advance. This can be done in $\mathcal{O}(n^2)$ time. A matlab example for this is shown in Listing 1.

Now that we know P we can use it to generate a uniform random sequence car by car. We just need to keep track of l and t and lookup the respective values of P we generated before. Again a matlab example for this is shown in Listing 2.

The algorithm shown in Listing 2 works the following way. For each car l we decide whether it is of a new or an old destination,

Listing 2: Matlab code to generate uniformly distributed random instances of length n

```
sequence(1)=1;
t=1;

for l=2:n
    if (rand < (t*P(t,l)/P(t,l-1)))
        sequence(l)=randi(t);
    else
        t=t+1;
        sequence(l)=t;
    end
end
```

i.e. of destination $t+1$ or in $\{1,\dots,t\}$. The probability to choose an old destination is calculated as

$$\frac{t \cdot P(t,l)}{P(t,l-1)}$$

i.e. the number of possibilities we would have after assigning an old destination to car l divided by the number of total possibilities there are now. In case we decide for one of the old destinations, we choose the car uniformly from $\{1,\dots,t\}$ (by Lemma 4.3) otherwise it is chosen to be of destination $t+1$. Note that once we have calculated the lookup-table for 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 is a partition of the set

$$\{1,2,\dots,n(S)\}$$

and the procedure above can generate any partition of this set with equal probability we now have a method at hand that uniformly draws a random partition of the set $\{1,\dots,n\}$. Also note that the number $P(1,1)$ from the pre-generated table (for example in listing 1) therefore gives the total number of possible set partitions of a set of size n. This means it is equal to the n-th Bell number B_n [14].

4.2 RESULTS

Now that we know how to generate the uniformly distributed problem instances we will perform some computational experiments. We will analyze the performance of the different bounds from Section 2.2, namely the upper bounds we get from Theorem 2.3 (in the figures denoted by n-UB) and Theorem 2.4 as

well as the lower bounds from Theorem 2.9 (in the figures denoted by u-LB) and Theorem 2.11 (in the figures denoted by Disjoint Sub-instance Bound - DSIB). In the last case the bound was calculated using Algorithm 3. Furthermore we will compare the performance of the two online algorithms from Section 3.3, i.e. the Split and the Unsplit Algorithm. Note that Unsplit and Theorem 2.4 return the same number of sorting tracks.

We will use sample instances of length $n = 50$ and $n = 100$. Figure 12 and Figure 13 each show the number of sorting tracks used by the algorithms or given by the bounds for 100 uniformly and independently chosen problem instances, as well as the optimal values $K(S)$.

Table 1 and 2 show in the second column the average results, i.e. our estimate for the expected result on instances from S^{50} and S^{100}. Each time the average was taken over 5000 independently generated random instances.

The third column shows the average ratio between the algorithms or the bounds output and $K(S)$, i.e. the value

$$\frac{\sum_{i=1}^{5000} \frac{\mathrm{Output}(S_i)}{K(S_i)}}{5000}$$

where $\mathrm{output}(S_i)$ is the respective output of the algorithm or bound for the i-th problem instance.

Finally in the fourth column 95% confidence intervals L are given for the results, i.e. we expect the actual value for the expected output to be no more than L away from the calculated average with 95% probability. The value L is calculated as

$$L = \Phi^{-1}(1 - \frac{0.05}{2}) \cdot \sqrt{\frac{V}{5000}}$$

where Φ^{-1} is the inverse of the cumulative distribution function of the normal distribution and V is the unbiased estimate for the variation obtained from the resulting data, i.e.

$$V = \frac{1}{5000 - 1} \sum_{i=1}^{5000} (\mathrm{Output}(S_i) - Avg)^2$$

where Avg is the mean value from the second column.

For more information on the underlying theory of estimating expectations and confidence intervals the reader may be referred to [12].

Observe that the two lower bounds provide results that are already very close to the optimum. On the other hand there is still a quite big gap between the optimal results and the upper bounds. Here might be an area where good heuristics can be applied in future work.

Note that Split an Unsplit perform almost equal (Split actually gives slightly better results) even though Split is not competitive and there exist problem instances such that Split returns arbitrarily bad results.

Also we can see that the upper bound obtained by Theorem 2.3 ($\lceil \frac{n(S)}{4} + \frac{1}{2} \rceil$) tends to strongly overestimate $K(S)$ in most cases. Therefore it might not be a good choice in application, except of course for the special instance classes, where it has a better performance (see Section 2.3.1).

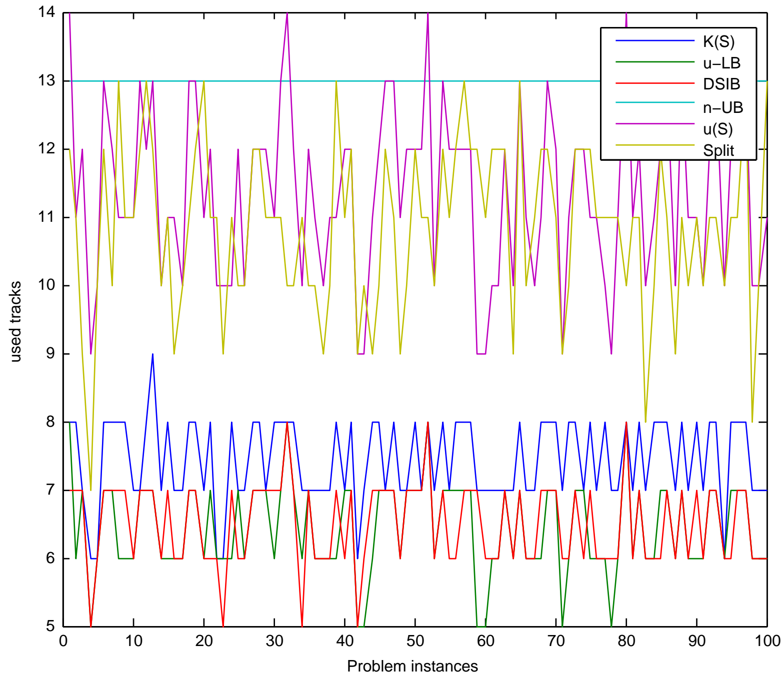


Figure 12: Comparing bound and algorithm results for 100 instances in S^{50}

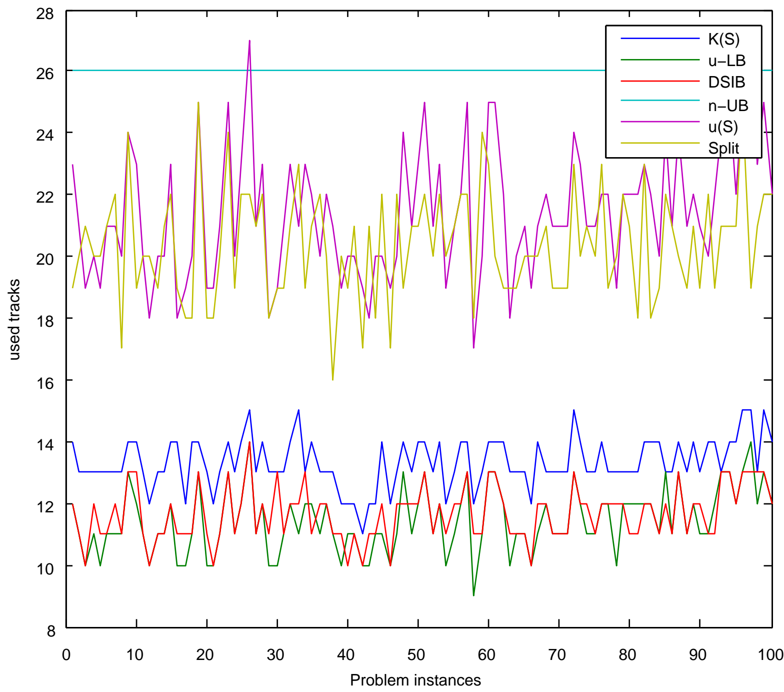


Figure 13: Comparing bound and algorithm results for 100 instances in S^{100}

	Mean	avg. ratio to $K(S)$	L
$K(S)$	7.4794	1	0.0184
$\lceil \frac{u(S)+1}{2} \rceil$	6.4984	0.8702	0.0206
DSIB	6.5892	0.8829	0.0176
$\lceil \frac{n(S)}{4} + \frac{1}{2} \rceil$	13	1.7521	0
$u(S)$	11.5086	1.5400	0.0388
Split	10.9096	1.4632	0.0375

Table 1: Average results over 5000 repetitions for different bounds and algorithms over uniform random instances of length $n = 50$

	Mean	avg. ratio to $K(S)$	L
$K(S)$	13.2580	1	0.0230
$\lceil \frac{u(S)+1}{2} \rceil$	11.3912	0.8596	0.0270
DSIB	11.5542	0.8721	0.0233
$\lceil \frac{n(S)}{4} + \frac{1}{2} \rceil$	26	1.9689	0
$u(S)$	21.2930	1.6064	0.0523
Split	20.3490	1.5370	0.0505

Table 2: Average results over 5000 repitions for different bounds and algorithms over uniform random instances of length $n = 100$

5 PROBLEM VARIATIONS

5.1 FIXED NUMBER OF TRACKS

Here we will consider the case when we only have a fixed number of tracks K available for marshalling (fixed tracks TMP). Obviously we can not expect to be able to sort all cars (except of course, if $K \geqslant K(S)$). So the goal will be to find a solution, such that a maximal number of destinations can be sorted completely, i.e. we allow destinations to be thrown away. Therefore we can define the optimal objective in the following way

Maximize the number of destinations that can be placed on a fixed number of sorting tracks

Definition 5.1. For any problem instance $S \in \mathcal{S}$ we denote with $T_K(S)$ the maximal number of destinations that can be sorted using K sorting tracks.

Using the results we already obtained for the original TMP we can now get corresponding results for the fixed tracks TMP. Again we will look at the offline as well as the online case.

5.1.1 *Offline Problem*

First note that as in the case of the classical TMP finding $T_K(S)$ is NP-hard in general. This is clear as otherwise we could easily determine the smallest K such that $T_K(S) = t(S)$, using bisection. But as before we can find a lower bound on the optimal objective value.

Fixed tracks TMP is NP-hard

Theorem 5.2. *For all $S \in \mathcal{S}$ and $K \leqslant u(S)$ we have*

$$T_K(S) \geqslant \lceil \frac{K \cdot t(S)}{u(S)} \rceil$$

Proof. Let $L = (L_1, \dots, L_{K(S)})$ be a solution for the classical TMP that was obtained in the way of Theorem 2.4, i.e. the cars are arranged corresponding to a coloring of the interval graph G^S. This also means that there are no split destinations. Now let

$$(n_i)_{i \in \{1,\dots,K\}}$$

be the indices of the K sorting tracks that hold the most destinations. Obviously sorting the cars as they are placed on the L_{n_i} is a solution to fixed tracks TMP, so we have

$$T_K(S) \geqslant \sum_{i=1}^{K} t(L_{n_i})$$

where $t(L_{n_i})$ is the number of destinations that is placed upon L_{n_i}. As we use the solution from the interval coloring approach we do not need to worry about split destinations in this case.

Furthermore the L_{n_i} obviously hold at least as many cars on average as the original $u(S)$ sorting tracks:

$$\frac{1}{K}\sum_{i=1}^{K} t(L_{n_i}) \geqslant \frac{1}{u(S)}\sum_{i=1}^{u(S)} t(L_i) = \frac{t(S)}{u(S)}$$

(the last equality holds as every destination appears on one of the sorting tracks of L). Multiplication with K now yields

$$T_K(S) \geqslant \sum_{i=1}^{K} t(L_{n_i}) \geqslant \left\lceil \frac{K \cdot t(S)}{u(S)} \right\rceil$$

where the ceiling function comes from the fact that the middle term must be an integer number. □

5.1.2 *Online Problem*

Again we can consider the corresponding online problem. But despite of the results we got in Chapter 3 for the original TMP we cannot get a competitive online algorithm for the variation with a fixed number of tracks. Not even when using randomization:

Theorem 5.3. *There is no competitive randomized online algorithm for fixed tracks TMP*

No online algorithm is competitive for fixed tracks TMP

Proof. In this proof we will fix the number of usable tracks to $K = 1$. Let alg be any randomized online algorithm for fixed tracks TMP and let $1 > p > 0$. Then construct a sequence of cars the following way:

Let $i = 1$. Now repeat the following steps n times:

- add a car with destination i.
- if alg places i on the sorting track with probability less than p then add a second and last car of destination i. We call those blocks coupled destinations.
- set $i = i + 1$.

Till now we call the sequence $T^{S'_n}$. We generate a first complete sequence $T^{S^1_n}$ by adding the remaining second cars that were not sent so far.

When given sequences of the type $T^{S^1_n}$ to alg there are two cases

1. case: There is no n such that alg takes any of the uncoupled destinations with probability greater or equal $1 - p$.

This means for any n there are at most $\frac{1}{p}$ uncoupled destinations. In this case alg sorts less than pn destinations on average while at least $n - \frac{1}{p}$ destinations (the coupled ones) are optimal. So for $T^{S^1_n}$ alg achieves a competitive ratio of more than

$$\frac{n - \frac{1}{p}}{pn} = \frac{1}{p} - \frac{1}{p^2 n} \overset{n \to \infty}{\to} \frac{1}{p}$$

2. case: There is such an n.
Now we generate another sequence based on $T^{S'_n}$:

$$T^{S^2_n} = (T^{S'_n}, n+1, n+1, n+2, n+2, \dots, m, m, *)$$

where $*$ stands for the missing second cars we also added at the end of $T^{S^1_n}$. Here sorting at least $m - (n+1)$ cars is optimal while alg achieves on average at most

$$p(m - (n+1)) + (1-p) + pn$$

leading to a competitive ratio of at least

$$\frac{m - (n+1)}{p(m - (n+1)) + (1-p) + pn} = \frac{1}{p + \frac{(1-p)+pn}{m-(n+1)}} \overset{m \to \infty}{\to} \frac{1}{p}$$

Therefore in both cases alg cannot be better than $\frac{1}{p}$ competitive. As we can choose p arbitrarily small in the beginning we get that alg is not competitive.

□

5.2 USING AN AUXILIARY TRACK

Now we will consider a modified version of the original TMP, where we are allowed to use an auxiliary track of capacity m on which cars can be stored before passing the switches. If a car is stored on the auxiliary track it can be rolled over the hump and sorted to the classification tracks at any later time, allowing for other cars to be sorted first. Of course the order of cars on this auxiliary track cannot be interchanged. That means the auxiliary track can be seen as some kind of first-in-first-out buffer. See figure 14 for schematic picture of this modified hump yard.

To see how the new track can affect the number of necessary tracks we will calculate some bounds based on the number of tracks used in the original problem. First we will show that an auxiliary track with infinite capacity will always (except for the trivial case with $K(S) = 1$) save at least one track.

Theorem 5.4. *For any TMP instance $S \in \mathbb{S}$ with $K(S) \geqslant 2$ we can save at least one track using an auxiliary track with capacity $m = \infty$.*

An auxiliary with ∞ capacity saves at least one track

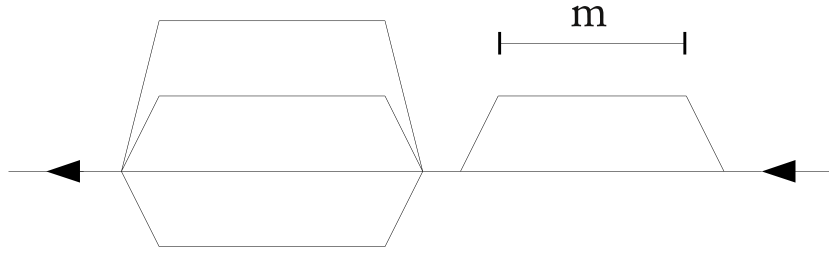


Figure 14: Schematic hump yard with auxiliary track of capacity m

Proof. Let $S \in \mathcal{S}$ be any TMP instance with $K(S) \geqslant 2$. Then in the final sorting for the original TMP let i be the last and j the second last classification track from which the cars will be pulled out.

Now we can place all the cars form i on the auxiliary track and place them on track j after all other cars where sorted. This way we only need $K(S) - 1$ tracks in total. □

Next we can show that an auxiliary track of infinite capacity can, in the best case, save an arbitrary amount of tracks.

An auxiliary track with ∞ capacity can save arbitrarily many tracks

Theorem 5.5. *For all $o \in \mathbb{N}$ there exists a sequence $S \in \mathcal{S}$ such that an auxiliary track with capacity $m = \infty$ will save at least o classification tracks.*

Proof. Let S be a sequence of cars with

$$T^S = (1, 2, \ldots, 2o + 1, 1, 2, \ldots, 2o + 1)$$

As $u(S) = 2o + 1$ we know from Theorem 2.9 that

$$K(S) \geqslant \lceil \frac{u(S) + 1}{2} \rceil = \lceil \frac{2o + 2}{2} \rceil = o + 1$$

Now using our modified model we can place the first $2o + 1$ cars on the auxiliary track. Now we can alternately place one car from the remaining sequence and one car from the auxiliary track on exactly one classification track. Therefore we saved at least o tracks. □

Next we consider the case where $m < \infty$. Allowing arbitrary instances in this case we can always find instances where the auxiliary track will have no effect on the optimal solution:

Theorem 5.6. *For any $m, k \in \mathbb{N}$ there exists a sequence of cars $S \in \mathcal{S}$ such that even using the auxiliary track of capacity m we will use at least $K(S) \geqslant k$ classification tracks.*

Proof. First we need a sequence $S' \in \mathcal{S}$ with $K(S') = k$. For example we can take

$$T^{S'} = (1, 2 \ldots, 2k - 1, 1, 2, \ldots, 2k - 1)$$

where as before Theorem 2.9 assures that $K(S') \geqslant k$.

Now we construct a new sequence S from S' by simply repeating every car $m+1$ times. Obviously this renders the auxiliary track useless as it only has capacity m but would need to store $m+1$ cars to have a positive effect on the sorting process. Therefore we will still need at least $K(S) \geqslant k$ tracks. □

Even though the auxiliary track may save a lot of sorting tracks, depending on the incoming sequence and the capacity of the auxiliary track, it might be a time consuming task to use the track. This is mainly because now some of the cars need to be placed on the auxiliary track first, while later the cars have to be brought to the sorting tracks from two different tracks. This way one might need many more additional operations than simply pushing all cars over the hump in one row.

5.3 MULTIPLE SORTING STEPS

Next we will consider the case, where we allow the incoming train to be sorted in multiple marshalling steps. This means we use the outgoing train of step i as the incoming train in step $i+1$. Obviously the outgoing train in every but the last step does not have to be sorted by destinations.

Defining the corresponding online problem in this case is not as easy as for the original TMP. For example one might consider the case, where all cars arrive before finishing the first marshalling step or the case where cars can arrive at any time throughout the sorting process. As it seems that there is no intuitive way to deal with this, we will not further discuss the online case of this problem variation.

Definition 5.7. For the TMP with multiple sorting steps let $K_q(S)$ be the minimal number of necessary tracks to sort S in q steps.

Of course we have $K_1(S) = K(S)$. Next we will show that multiple sorting steps can reduce the number of necessary tracks by an approximate factor equivalent to the number of steps:

Theorem 5.8. *For all $S \in \mathcal{S}$ and $q \in \mathbb{N}$ we have*

$$K_q(S) \leqslant \lceil \frac{K(S)-1}{q} \rceil + 1$$

q sorting steps can reduce tracks by approximately a factor q

Proof. This can be shown by constructing a valid solution for the multiple stage problem based on an optimal solution from the original TMP. Let $L_1, \ldots, L_{K(S)}$ be the sorting tracks necessary to optimally solve S.

For our new solution we want to use only tracks $L'_1, \dots, L'_{o+1}$, with

$$o = \lceil \frac{K(S) - 1}{q} \rceil$$

In the first step we will place the cars from L_i on track L'_i for all $i \in \{1, \dots, o\}$. All other cars will be stored on track L'_{o+1}. The outbound train of this step will consist of a sorted part in the beginning and an unsorted part at the end. Now in the following steps we can continue accordingly. In step j we first place the already sorted part of the incoming train on L'_1. Afterwards all cars from $L_{i+(j-1)\cdot o}$ will be placed on L'_i for all $i \in \{1, \dots, o\}$. Again all remaining cars will be placed on L'_{o+1}. As there will be no remaining cars in step q, track L'_{o+1} can also be used for sorting. So in total we can handle

$$q \cdot o + 1 = q \lceil \frac{K(S) - 1}{q} \rceil + 1 \geqslant K(S)$$

of the original sorting tracks, therefore arriving at a feasible solution for the multiple sorting steps problem. □

Example 5.9. Figure 15 shows an example how a problem instance can be sorted in the way of Theorem 5.8. The problem instance is a pair problem instance with maximal overlapping destinations. Therefore Algorithm 4 will generate an optimal solution that uses $K(S) = 7$ sorting tracks. Using the procedure of Theorem 5.8 we only need

$$\lceil \frac{7 - 1}{2} \rceil + 1 = 4$$

sorting tracks.

Using multiple sorting steps can be a good way to deal with a constraint on the total number of sorting tracks available. In reality it might be simply impossible to sort a train according to the TMP model as for a certain problem instance S there might not be $K(S)$ sorting tracks that are usable. In this case one might want to take a solution that works with a reduced number of sorting tracks - and is therefore feasible in reality - but needs more sorting steps. Here one accepts the increased time necessary for the additional couplings and decouplings.

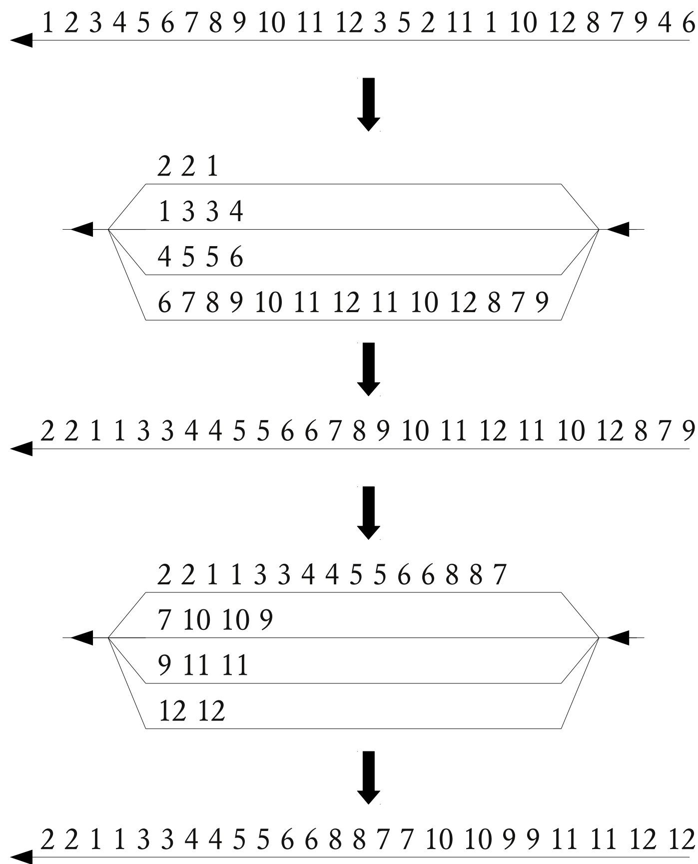


Figure 15: Example for multiple sorting steps

6 CONCLUSIONS

In this thesis we have taken a closer look at the train marshalling problem. Regarding the fact that solving TMP in general is NP-hard (see Theorem 2.1) we state some lower and upper bounds on the optimal objective value. In this thesis we derive a new lower bound using graph theoretical properties of the problem instance (see Theorem 2.11). The bound can be calculated in polynomial time by searching for maximal cliques in subgraphs of the problem as it is done by Algorithm 3.

Next we looked at problem instances and problem restrictions that can be solved in polynomial time. On the one hand we have instances for which one of the lower and upper bounds are equal. Here we could derive fast algorithms to solve the instances optimally. On the other hand we showed that fixing the number of destinations or the number of overlapping destinations makes it possible to solve the problem in polynomial time (though not necessarily fast).

Furthermore we considered the online version of TMP. Here we can show that the best possible deterministic online algorithm (Algorithm 6) is based on a simple interval coloring scheme and uses at most two times as many sorting tracks as necessary.

When comparing the results from Chapter 4 we saw that the two lower bounds from Section 2.2 tend to be quite close to the optimal value $K(S)$, while there is still some gap to the upper bounds.

In the last chapter we looked at some variations of TMP and gave some bounds on their objective values. As the variations are closely related to the original problem we get bounds that are based on the results we got in chapter 2. For example we found out that there is no competitive online algorithm for the problem with a fixed number of sorting tracks and that using multiple sorting steps leads to a reduced number of sorting tracks in most of the cases.

6.1 OPEN QUESTIONS

There are several question that still remain unanswered and that might be of interest for future research.

- What complexity does TMP have, if we fix or bound the number of cars per destination? Throughout this thesis we have often worked with pair problem instances, but still we do not know if a restriction to this problem class can

be solved in polynomial time or if there is an NP-hardness proof for it.

- In the online case of TMP one might be interested in a good lower bound for randomized online algorithms. Showing lower bounds in this case is much more difficult than in the deterministic case, as one can not be sure on the choices of the algorithm when constructing a bad input sequence. Therefore it would be interesting to know if there is a randomized online algorithm that is strictly better than 2-competitive. See also Section 3.4.

- It might be interesting to see if the auxiliary track from Section 5.2 can improve the lower bound on deterministic online algorithms for TMP, i.e. if there is a better than 2-competitive deterministic online algorithm that utilizes an auxiliary track.

- One might wish for better results in the multiple sorting step variation from Section 5.3. For example it would be interesting to know if there is a better bound than in Theorem 5.8 or if one can generate sequences for which it is always sharp.

- In Chapter 4 we used uniformly distributed instances to get an idea about the average case performance of the different bounds and algorithms. It would be interesting to see the performance on different sets of real world data.

BIBLIOGRAPHY

[1] Katharina Beygang. personal communication. to appear in Katharina Beygang's dissertation.

[2] Ulrich Blasum, Michael R. Bussieck, Winfried Hochstättler, Christoph Moll, Hans-Helmut Scheel, and Thomas Winter. Scheduling trams in the morning. *Mathematical Methods of Operations Research*, 49:137–148, 1999. ISSN 1432-2994. URL http://dx.doi.org/10.1007/s001860050018. 10.1007/s001860050018.

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

[4] Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. Robust algorithms and price of robustness in shunting problems. In *Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization and Systems (ATMOS)*, pages 175–190, 2007.

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

[6] Elias Dahlhaus, Fredrik Manne, Mirka Miller, and Joseph F. Ryan. Algorithms for combinatorial problems related to train marshalling. *AWOCA*, pages 7–16, 2000.

[7] Michael Gatto, Jens Maue, Matúš Mihalák, and Peter Widmayer. Shunting for dummies: An introductory algorithmic survey. In Ravindra Ahuja, Rolf Möhring, and Christos Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 310–337. Springer Berlin / Heidelberg, 2009. URL http://dx.doi.org/10.1007/978-3-642-05465-5_13. 10.1007/978-3-642-05465-5_13.

[8] U. I. Gupta, D. T. Lee, and J. Y.-T. Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12:459–467, 1982.

[9] Donald Ervin Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley Publishing Company, 1973.

[10] Christian Liebchen, Marco Lübbecke, Rolf Möhring, and Sebastian Stiller. The concept of recoverable robustness, linear

programming recovery, and railway applications. In *Robust and Online Large-Scale Optimization*, pages 1–27. Springer, Heidelberg, 2009.

[11] Christian Lindecke. Ablaufberg.png, 2009. URL http://upload.wikimedia.org/wikipedia/commons/8/8f/Ablaufberg.png.

[12] Thomas Müller-Gronbach, Erich Novak, and Klaus Ritter. *Monte Carlo-Algorithmen*. Springer-Lehrbuch. Springer, 2011.

[13] Stephan Olariu. An optimal greedy heuristic to color interval graphs. *Information Processing Letters*, 37(1):21–25, 1991. ISSN 0020-0190. doi: DOI:10.1016/0020-0190(91)90245-D. URL http://www.sciencedirect.com/science/article/B6V0F-45GMFHF-2J/2/7fbdc1de385b1809910f81e6c9e21deb.

[14] Gian-Carlo Rota. The number of partitions of a set. *The American Mathematical Monthly*, 71(5):pp. 498–504, 1964. ISSN 00029890. URL http://www.jstor.org/stable/2312585.

[15] M.W. Siddiqee. Investigations of sorting and train formation schemes for a railroad hump yard. In *Proceedings of the 5th International Symposium on the Theory of Traffic Flow and Transportation*, pages 377–387, 1972.

[16] Gabriele Di Stefano and Magnus Love Koci. A graph theoretical approach to the shunting problem. *Electronic Notes in Theoretical Computer Science*, 92:16–33, 2004. ISSN 1571-0661. doi: DOI:10.1016/j.entcs.2003.12.020. URL http://www.sciencedirect.com/science/article/B75H1-4BY45WP-F/2/49e2cf3a8c99ed487b5ac1a0e3e395bb. Proceedings of ATMOS Workshop 2003.

DECLARATION

I hereby declare that I am the only author of this work and that no other sources than those listed have been used.

Kaiserslautern, December 2010

Florian Dahms