SHUNTING OF TRAMS IN STORAGE YARDS

1. INTRODUCTION

For railway optimisation we have to consider a lot of parameters. The lines of the railway network have to be planned, timetables generated, personnel and stock have to be scheduled, shunting has to be avoided, and many more.

In the following we focus on the aspect “shunting”. Intuitively we have a more or less detailed picture in mind what happens if trains have to be shunted. The topic shunting covers the routing of train units, the scheduling of personnel, cleaning and maintenance of vehicles, and the parking of train units. Of course we are always interested to have as less shunting movements as possible. We are going to look into the last part focusing on the dispatch of trams in storage yards.

The background situation is a storage yard where the trams are stored one behind another in dead-end sidings we are going to consider as stacks. Due to the lack of space arriving trams have to be assigned to a depot position immediately even if the arrival sequence of the following trams is not known completely. Thus real-time dispatch of trams in storage yards is a typical online problem.

We are going to model an optimal offline problem called Minimal Shunting Problem (MSP). Then we will give the corresponding linear integer program of (MSP) and have a look at two online algorithms. Finally we will show that there is no online algorithm of (MSP) where the number of shunting movements can be bound by an independant constant.

2. BASIC NOTATIONS

Before we can describe mathematical models for the (MSP) we need some basic notations. In a storage yard with $R$ dead-end sidings each siding corresponds to a stack. The positions in each stack are denoted by $P_r$ for $1 \leq r \leq R$. The positions are numbered consecutively from bottom to top starting in stack one. The set of all positions is denoted by $\mathcal{P} = \{1, \ldots, P\}$. Let $\mathcal{A} = \{1, \ldots, N\}$ be the set of all trams and $\mathcal{D} = \{1, \ldots, M\}$ denote the set of all departures. We identify a tram with its index number in the sequence
of arrivals at the depot while a round trip is identified with its index number in the sequence of required departures. We assume that there are different types of trams and denote the set of all tram types by $T = \{1, \ldots , T\}$. Then we assign a type to a tram by $t_a: A \to T$ and a type required for a round trip by $t_d: D \to T$.

In the following we assume $M = N = P$, that is the depot is empty before the first tram arrives and there are as many positions as trams. We also assume that the number of trams of one type coincides with the number of round trips requiring this type of trams:

$$\#\{i \in A| t_a(i) = \tau\} = \#\{j \in D| t_d(j) = \tau\}$$

for each $\tau \in T$.

To assign the trams to the positions in the stack we use a binary variable $x_{iq}$ where

$$x_{iq} := \begin{cases} 1 & \text{if tram } i \text{ is assigned to position } q \\ 0 & \text{else.} \end{cases}$$

Analogously we define the assignment at departure, where

$$y_{jq} := \begin{cases} 1 & \text{if departure } j \text{ is assigned to position } q \\ 0 & \text{else.} \end{cases}$$

The complete assignments are represented by the bijective mappings

$$\pi_X: A \to P \quad \text{and} \quad \pi_Y: D \to P.$$ 

**Remark 1.** $\pi_X(i) = q$ and $\pi_Y(j) = q$ if and only if $x_{iq} = y_{jq} = 1$.

Now we can assign both arriving trams and their departures to positions in the stacks. As mentioned above not all trams are of the same tram type. But of course it is very important to be sure that a departure is assigned to a tram of the right type. Thus the assignment for the arriving tram and for the departure concerning the same stack position have to yield the same tram type.

Therefore we formulate the following

**Definition 1.** Two assignments $\pi_X: A \to P$ and $\pi_Y: D \to P$ are called type preserving if and only if $t_a(\pi_X^{-1}(q)) = t_d(\pi_Y^{-1}(q))$ for all positions $q \in P$.

For $i \in A$ and $j \in D$ we introduce $\theta_{ij}$ where

$$\theta_{ij} := \begin{cases} 1 & \text{if } t_a(i) = t_d(j) \\ 0 & \text{else} \end{cases}$$

For $\pi_X(i) = \pi_Y(j) =: p$ and $\theta_{ij} = 0$ we observe a type mismatch at position $p$.

Two trams $i, j$ with $i < j$ have to be shunted after arrival if they are assigned to positions $q, p$ in the same stack $P_r$ with $x_{iq} = x_{jp} = 1$ and $q > p$. 


**Figure 1. Example for a type-mismatch**

Analogously the two trains have to be shunted if the departure of the train at position $p$ is planned before the departure of the train at position $q$; i.e. $y_{kq} = y_{lp} = 1$ and $l > k$.

**Remark 2.** Trains $i,j$ ($i < j$) have to be shunted after arrival if and only if $\pi_X(i) = q < p = \pi_Y(j)$, $q,p \in P_r$, for some $r \in \{1, \ldots, R\}$.

Analogously trains $k,l$ ($k < l$) have to be shunted at departure if and only if $\pi_Y(k) = q < p = \pi_Y(l)$, $q,p \in P_r$, for some $r \in \{1, \ldots, R\}$.

We can describe these shunting movements on arrival by the coefficients $\alpha_{ij}$ and $\beta_{qp}$, where

$$
\alpha_{ij} := \begin{cases} 
1 & \text{if } i < j \\
0 & \text{else}
\end{cases}
$$

$$
\beta_{qp} := \begin{cases} 
1 & \text{if } p < q \\
0 & \text{else}
\end{cases}
$$

Again the shunting movements at departure can be described analogously by the coefficients $\alpha'_{kl}$ and $\beta'_{pq}$, where

$$
\alpha'_{kl} := \begin{cases} 
1 & \text{if } k < l \\
0 & \text{else}
\end{cases}
$$

$$
\beta'_{pq} := \begin{cases} 
1 & \text{if } q < p \\
0 & \text{else}
\end{cases}
$$

**Remark 3.** Trains $i,j$ with $i < j$ have to be shunted on arrival if and only if

$$
\alpha_{ij} \beta_{qp} x_i x_j p = 1.
$$

Similarly these trains have to be shunted at departure if and only if

$$
\alpha'_{kl} \beta'_{pq} y_k y_l p = 1.
$$

**Definition 2.** We call two assignments $\pi_X$ and $\pi_Y$ shunting-free if no shunting movements occur both on arrival and at departure.
Figure 2. Example for an assignment that requires shunting at departure

3. Minimum Shunting Problem

We now focus on the Minimum Shunting Problem (MSP). That means we want to minimize the number of shunting movements on arrival and at departure for some type preserving assignments $\pi_X$ and $\pi_Y$ we introduced in the previous section.

We get the following binary program (MSP):

$$\min \sum_{q \in P} \sum_{p \in P} \left( \sum_{i \in A} \sum_{j \in A} \alpha_{ij} \beta_{pq} x_{iq} x_{jp} + \sum_{j \in D} \sum_{k \in D} \alpha_{jk} \beta_{pq} y_{jq} y_{kp} \right)$$

subject to:

$$\sum_{i \in A} x_{iq} = 1 \text{ for all } q \in P$$

$$\sum_{q \in P} x_{iq} = 1 \text{ for all } i \in A$$

$$\sum_{j \in D} y_{jq} = 1 \text{ for all } q \in P$$

$$\sum_{q \in P} y_{jq} = 1 \text{ for all } j \in D$$

$$x_{iq} + y_{jq} \leq 1 \text{ for all } i \in A, j \in D, q \in P, \theta_{ij} = 0$$

$$x_{iq} \in \{0, 1\} \text{ for all } i \in A, q \in P$$

$$y_{jq} \in \{0, 1\} \text{ for all } j \in D, q \in P$$
The first four constraints make sure that each tram can occupy exactly one parking slot in the depot. Constraint number five forces the assignments to be type preserving, while the last two conditions refer to the binary variables $x$ and $y$.

As we minimize over a subset of $N$, this problem has an optimal solution with a minimum number of shunting movements at arrival and departure. For practical reasons we are interested to reduce all the shunting either to the arriving part or to the departure.

**Theorem 1.** (MSP) has an optimal solution without shunting either at arrival or at departure.

**Proof.** Due to symmetry it suffices to prove the existence of an optimal solution without shunting at departure. Let $\pi_X : A \rightarrow \mathcal{P}$ and $\pi_Y : D \rightarrow \mathcal{P}$ denote two type preserving assignments for (MSP). And let $\sigma_1 : \mathcal{P} \rightarrow \mathcal{P}$ and $\sigma_2 : \mathcal{P} \rightarrow \mathcal{P}$ denote two permutations, where $\sigma_1 = \tau_1 \cdot \ldots \cdot \tau_r$, $\sigma_2 = \tau_2 \cdot \ldots \cdot \tau_s$, the $\tau_j$ transpositions, with $\sigma_1 \pi_X$ and $\sigma_2 \pi_Y$ are two type-preserving assignments for an optimal solution of (MSP) and the equation $t_d(\pi_X^{-1}(\sigma_1(q))) = t_d(\pi_Y^{-1}(\sigma_2(q)))$ holds. As $\sigma_2$ is bijective we can write $q = \sigma_2^{-1}(q')$. Thus $t_d(\pi_X^{-1}(\sigma_1(\sigma_2^{-1}(q')))) = t_d(\pi_Y(q'))$ for all $q' \in \mathcal{P}$. Since with $\sigma_1 \sigma_2^{-1} = \tau_1 \cdot \ldots \cdot \tau_r \cdot \tau_2 \cdot \ldots \cdot \tau_s$ the total number of transposition does not change we now have an optimal solution without shunting at departure. $\square$

4. **(MSP) Applied to a Queue**

We have modelled the (MSP) for storage yards with dead-end sidings we considered as stacks. That holds for the railway system, the dispatch of trams, and any other transport system where the vehicles can easily change their direction from forwards to backwards. But for example this model is not convenient for the bus transport system. It would take a lot of time to turn long buses round every time they start a new trip. Thus in this situation the storage yards do not resemble a stack but a queue and therefore we would like to apply the (MSP) for queues, too. Looking at the modeling we see, that the only part the property of a stack was used was the introduction of the coefficients $\alpha'_{ij}$ and $\beta'_{pq}$ describing shunting movements at departure. Hence we have to change the coefficient $\beta'_{pq}$ to $\beta''_{qp}$, where

$$\beta''_{qp} := \begin{cases} 1 & \text{if } p < q \\ 0 & \text{else.} \end{cases}$$

That suffices to make the model applicable to a situation where we are dealing with queues instead of stacks.

5. **Modifications**

For modeling the (MSP) we assumed that there are exactly as many trams as positions in the stacks, that all arriving trams will stay at the depot until
the next scheduling period begins, and that all trams have the same length. In general those constraints will be too tight.

First, there may be more positions than trams and there may be less departures than trams. Thus it may happen that the depot is empty before the first arrival. Then the constraints

\[ \sum_{i \in A} x_{iq} = 1 \quad \text{and} \quad \sum_{j \in D} y_{jq} = 1 \]

can be relaxed to

\[ \sum_{i \in A} x_{iq} \leq 1 \quad \text{and} \quad \sum_{j \in D} y_{jq} \leq 1 \]

As we can only assign departures to positions containing a tram, i.e. \( \pi_y(D) \subset \pi_X(A) \), the second inequality is a direct consequence of the first one.

Furthermore we can assume that some trams stay at the depot after the departure of all trams from \( D \). That may be the case at weekends when not all lines are served. Then the remaining trams are added as dummy arrivals to the next arrival set \( A \). We have to note that dummy arrivals proceed all other arrivals. Also dummy arrivals will never be shunted with other dummy arrivals, therefore we set \( \alpha_{ij} := 0 \) for all dummy arrivals \( i, j \in A \).

Of course the assumption that all trams have the same length is far from reality. To overcome this let \( l_i \in \mathbb{N} \) denote the length of a tram and \( L_r \in \mathbb{N} \) the length of the \( r \)-th stack. Then

\[ \sum_{i \in A} \sum_{q \in P_r} l_i x_{iq} \leq L_r \quad \text{for stack } r. \]

6. **Linear Integer Program (MSPLIN)**

In order to find solutions we would like to apply certain solvers like the CPLEX. Unfortunately these can only deal with linear programs, thus we have to try to get rid of the quadratic terms.

As we have already observed, there is an optimal solution where the shunting is reduced to a minimum. Due to theorem 1 we can even restrict the set of solutions of (MSP) by adding the following constraint which enforces solutions which are shunting-free at departure:

\[ y_{iq} + y_{kl} \leq 1 \quad \text{for all } j, k \in D, \ q, l \in P \quad \text{and} \quad \alpha_{ij} \delta_{lq} = 1. \]

By linearising the resulting tightened quadratic assignment problem using the method of Kaufman and Broeckx [4] we obtain the following linear integer program called (MSPLIN):

\[ \min \sum_{i \in A} \sum_{q \in P} z_{iq} \]
subject to:
\[
\sum_{i \in A} x_{iq} = 1 \text{ for all } q \in P
\]
\[
\sum_{q \in P} x_{iq} = 1 \text{ for all } i \in A
\]
\[
\sum_{j \in D} y_{jq} = 1 \text{ for all } q \in P
\]
\[
\sum_{q \in P} y_{jq} = 1 \text{ for all } j \in D
\]
\[
s_{iq} + d_{iq} x_{iq} - z_{iq} \leq d_{iq} \text{ for all } i \in A, q \in P
\]
\[
x_{iq} + y_{jq} \leq 1 \text{ for all } i \in A, j \in D, q \in P, \theta_{ij} = 0
\]
\[
y_{jq} + y_{kl} \leq 1 \text{ for all } j, k \in D, q, l \in P : \alpha_{ij} \beta_{iq} = 1
\]
\[
x_{iq} \in \{0,1\} \text{ for all } i \in A, q \in P
\]
\[
y_{jq} \in \{0,1\} \text{ for all } j \in D, q \in P
\]

where
\[
s_{iq} = \sum_{j \in A} \sum_{p \in P} \alpha_{ij} \beta_{qp} x_{jp} \text{ and } d_{iq} = \sum_{j \in A} \sum_{p \in P} \alpha_{ij} \beta_{qp}.
\]

We know that in an optimal solution of (MSP) a pair of trains of the same type in the same stack will never require shunting at arrival. For computational reasons we add the respective redundant constraints:
\[
x_{iq} + x_{jl} \leq 1 \text{ for all } i, j, q, l \in P : \alpha_{ij} \beta_{ql} = 1, \theta_{ij} = 1
\]

And we also replace the type constraints
\[
x_{iq} + y_{jq} \leq 1 \text{ for all } i \in A, j \in D, q \in P, \theta_{ij} = 0
\]

by the equivalent constraint set:
\[
\sum_{i \in A : \theta_{i \tau} = 1} x_{iq} - \sum_{j \in D : \theta_{j \tau} = 1} y_{jq} = 0 \text{ for all } q \in P, \tau \in T.
\]

Table 1 shows real-world data of storage yards at Braunschweig ("ba.*") and Karlsruhe ("ka.*"). Using the CPLEX 6.5 MIP-Solver Winter and Zimmermann [1] solve the modified integer program, where all computational results are obtained on a Pentium II PC with 350 MHz and 256 MByte core memory.

As we can see the time limit of one hour CPU-time was to small for the instances ka.26, ka.28 and ka.29. There the interesting point is that the number of trains and stacks are only twice the others and the constraints and nonzeros only become about four times as much as the others. Still the CPU-time grows in an immense height. Thus for the storage yards of
Table 1. Real-world data of storage yards at Braunschweig and Karkruhe

<table>
<thead>
<tr>
<th>instance</th>
<th>( N )</th>
<th>( R )</th>
<th>variables</th>
<th>constraints</th>
<th>nonzeros</th>
<th>MSP</th>
<th>CPU sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>bs.mo.do</td>
<td>27</td>
<td>9</td>
<td>1872</td>
<td>10062</td>
<td>33381</td>
<td>0</td>
<td>2857</td>
</tr>
<tr>
<td>bs.fr</td>
<td>27</td>
<td>9</td>
<td>1872</td>
<td>10062</td>
<td>33381</td>
<td>0</td>
<td>2857</td>
</tr>
<tr>
<td>bs.sa</td>
<td>27</td>
<td>9</td>
<td>1872</td>
<td>10062</td>
<td>33381</td>
<td>0</td>
<td>2857</td>
</tr>
<tr>
<td>bs.so</td>
<td>14</td>
<td>5</td>
<td>483</td>
<td>1149</td>
<td>4355</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ka.26</td>
<td>46</td>
<td>15</td>
<td>5424</td>
<td>60896</td>
<td>182744</td>
<td>(5)</td>
<td>&gt; 14400</td>
</tr>
<tr>
<td>ka.27</td>
<td>23</td>
<td>7</td>
<td>1311</td>
<td>6704</td>
<td>22738</td>
<td>0</td>
<td>945</td>
</tr>
<tr>
<td>ka.28</td>
<td>44</td>
<td>15</td>
<td>4939</td>
<td>47472</td>
<td>145963</td>
<td>(1)</td>
<td>&gt; 14400</td>
</tr>
<tr>
<td>ka.29</td>
<td>46</td>
<td>15</td>
<td>5422</td>
<td>60894</td>
<td>182737</td>
<td>(8)</td>
<td>&gt; 14400</td>
</tr>
</tbody>
</table>

bigger towns even the (MSPLIN) cannot provide a solution within a realistic time interval. Therefore we have to look for algorithms that can solve the problem in shorter time.

7. Heuristics

As we have noted above the times required for solving even small instances of (MSP) are much larger than the real-time bounds required in practical applications. Therefore we have to look for applications that can be used in real-time situations even though they might be suboptimal. Furthermore, the trains' arrival sequence is not known completely as side conditions such as traffic or others cannot be foreseen and we have to decide immediately on arrival which stack we take.

In the following section we will introduce two examples of heuristical approaches for (MSP), namely Last-In-First-Out (LIFO) and Best-Fit (BF). Based on those other approaches can be done. These applications assign the arriving trains only based on the knowledge of the departure sequence.

Firstly we introduce an acceptance parameter \( \Delta \).

Definition 3. The departure and the corresponding stack are accepted if for this assignment no shunting is required and the top element's departure differs from the train's departure by not more than \( \Delta \).
The departure and the corresponding stack are not accepted if for this assignment no shunting is required and the top element's departure differs from the train's departure by more than \( \Delta \).

Algorithm 1 (LIFO).

1. Assign tran \( i \) to the latest unassigned departure \( j \in D \) of the same type.
2. If shunting is required or the stack is not accepted and
   (a) there still is an empty stack, then assign tran \( i \) to this stack
   (b) there is no empty stack, then assign tran \( i \) to a stack for which the top element's departure has least distance to \( j \).
The idea is to fill the stacks from bottom to top with the later departures first. Furthermore we try to have small time-intervals between to consecutive elements. If there is no optimal solution we are interested in keeping the time-intervals narrow, even if shunting will be necessary. Otherwise we might avoid shunting in this particular case but cause a lot more shunting later.

Algorithm 2 (BF). Assign tram \(i\) to an unassigned departure \(j_0 \in \mathcal{D}\) of the same type such that \(j_0\) minimizes the distance to the top element’s departure \(j_r\) of a stack \(r\) in the set of all \(j\) where \(j \leq j_r\).

1. If such a \(j_0\) exists and
   (a) is accepted, then assign \(i\) to this departure and to this stack.
   (b) is not accepted, and
      (i) there is an empty stack, assign \(i\) to this stack and to the latest unassigned departure of this type.
      (ii) there is no empty stack, ignore the assignment parameter \(\Delta\) and assign \(i\) to the departure \(j_0\) and to stack \(r\).

2. If no such \(j_0\) exists and
   (a) there is an empty stack, then assign \(i\) to this stack and to the latest unassigned departure of this type.
   (b) there is no empty stack, then assign tram \(i\) to an unassigned departure \(j_0 \in \mathcal{D}\) of the same type and to a stack \(r\) such that \(j_0\) minimizes the distance to the top element’s departure \(j_r\) in the set of all \(j\) where \(j > j_r\).
The actual arrivals of trams may differ substantially from the scheduled arrivals. Since the information on the future arrival sequence is not known when assigning the current tram real-time tram dispatch is a typical online problem.

The performance of an online algorithm is usually measured by means of competitive analysis.

Let \( ALG \) be an online algorithm. Thus \( ALG \) receives the arrival sequence \( A \) of the (MSP) only tram by tram and has to assign it immediately without (complete) information on future arrivals. Thus the number of resulting shunting movements \( s_{ALG}(MSP) \) will be at least as large as the minimum number of shunting movements \( s \) of the (offline) (MSP).

**Definition 4.** An online algorithm \( ALG \) is \( c \)-competitive if for all instances (MSP) the inequality

\[
s_{ALG}(MSP) \leq c \cdot s(MSP) + b
\]

holds, where \( b \) is some constant independent of the instance (MSP).
In the following we will show that no online algorithm for the (MSP) is c-competitive for any c. We will describe an unbounded class of instances which admit shunting-free assignments, i.e. $s(MSP) = 0$. But any arbitrary (deterministic or randomized) online algorithm can be forced to construct assignments which require shunting movements which are linear in the number of trams. Thus we cannot find constants $c, b$ as required in definition 4.

**Proposition 1.** For the (MSP) no online algorithm is c-competitive for any c.

**Proof.** As shown in theorem 1 it suffices to restrict shunting to departure. We have an instance consisting of $N = 6m$ trams ($m \geq 2$) partitioned into 3 groups of $2m$ tram types denoted by $\{b, w, y\}$ (e.g. black, white and yellow trams). The trams have to be stored in 2 stacks of length $4m$ and $2m$. The known departure sequence consists of six groups of size $m$ each of which requires only one tram type

$$(b, \ldots, b, w, \ldots, w, y, \ldots, y, y, \ldots, y, b, \ldots, b, w, \ldots, w).$$

The first arriving tram is of type $b$. ALG has two possibilities to assign this tram: either to the bottom of stack 1, that is the one with length $4m$, or to the bottom of stack 2 which has length $2m$. Depending on its choice we define the full arrival sequence in order to get a shunting-free solution for the offline algorithm and the optimal solution an online algorithm can provide needs at least $2m$ shunting movements. Now, if ALG assigns the first black tram to stack 1, we define its full arrival sequence

$$A_1 \equiv (b, \ldots, b, b, \ldots, b, w, \ldots, w, y, \ldots, y, y, \ldots, y, w, \ldots, w).$$

Otherwise the full arrival sequence will be

$$A_2 \equiv (b, \ldots, b, y, \ldots, y, y, \ldots, y, b, \ldots, b, w, \ldots, w, w, \ldots, w).$$

The following figures show the two arrival sequences, the best assignments an online algorithm can provide, and the respective shunting-free solution an offline algorithm computes.

Let us start with the arrival sequence $A_1$, that is, the online algorithm has assigned the first black tram to stack 1. Now ALG has to assign all remaining $2m - 1$ black trams at the beginning. We already know that $m$ black trams are supposed to leave before all others. Thus these $m$ trams have to be shunted with at least one white or yellow tram. In order to avoid shunting with more than one of these trams we must assign all remaining $2m - 1$ black trams to stack 2. The last $m$ trams in $A_1$ are white. But as $m$ white trams have to leave after all the other trams we should assign one white tram to the top position of stack 2 in order to avoid at least one shunting movement with the black tram at the bottom of stack 1. Unfortunately
the remaining \( m - 1 \) black trams in stack 2 cause another \( m - 1 \) shunting movements. Thus we have at least \( m + (m - 1) = 2m - 1 \) shunting movements for an optimal solution of \( ALG \) and therefore we cannot provide a constant \( b \) such that \( s_{ALG}(MSP) \leq c \cdot s(MSP) + b \) holds.

Now we focus on \( A_2 \) where \( ALG \) has assigned the first black tram to the bottom position of stack 2. Here all white trams arrive at the end; thus these trams will be assigned to the \( 2m \) top most positions of stacks 1 and 2. We can also state that at least one white tram is assigned to stack 1.
Would stack 2 be filled up with black trams then all 2m white trams and all 2m yellow trams have to be assigned to stack 1. Therefore 2m yellow trams have to pass m white trams (they have to leave latest) causing 2m² shunting movements. As long as all white trams are assigned to stack 1 this number of shunting movements cannot be reduced. In the end at least one white tram is on top of the 2m black and 2m yellow trams.

We also know that m black trams have to leave at the beginning. If there is more than one white tram on top of both stacks at least 2m shunting movements are necessary. Let exactly one white tram be assigned to stack 2 then m-1 white trams in stack 1 have to be passed by 2m+1 black and yellow trams. If there is exactly one white tram in stack 1 then the black bottom tram in stack 2 passes m white trams and in stack 1 m black trams have to pass the white tram on top.

In any case we have at least 2m shunting movements and cannot find a constant b such that \( s_{ALG}(MSP) \leq c \cdot s(MSP) + b \) holds.

For any algorithm we have found an arriving sequence such that according to the algorithm’s first decision whether to assign the arriving tram in stack 1 or 2 the optimal solution that can be found for all the other arriving trams needs at least 2m shunting movements, i.e. it depends linearly on the number of trams.

\[ \square \]

References


