Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)

Moving Trains like Pebbles: A Feasibility Study on Tree Yards

Issa K. Hanou, Mathijs M. de Weerdt, Jesse Mulderij

Delft University of Technology
i.k.hanou@tudelft.nl
m.m.deweerdt@tudelft.nl

Abstract

The Train Unit Shunting Problem concerns the parking of trains outside their scheduled use on so-called shunting yards. This is an NP-hard problem, and the current algorithm used by the Netherlands Railways cannot detect whether an instance is infeasible. So, infeasible instances can cause needlessly long computation times. Therefore, this paper fills the gap by providing novel approaches to determine the feasibility. For this, the Pebble Motion problem is considered which moves pebbles from their starting node to their goal node in the graph, such that no two pebbles occupy a node at the same time. A variant of the Pebble Motion problem is proposed to model the Train Unit Shunting Problem, where train units are represented by pebbles and the arrival and departure of train unit combinations are also included. This paper specifically looks at dead-end track shunting yards, as they can be abstractly represented by trees, such that trains arrive and depart at the root node. Furthermore, trains cannot be reallocated between arrival and departure in the tree, since reallocation in practice is a very costly process as moves need to be performed by a small set of drivers. The conditions for realizing the departure order of trains are studied, and an efficient method to (partially) determine the feasibility of problem instances is given, which can find the minimal number of tracks required to park the trains. Furthermore, a special case with tracks of length two is shown to be polynomially solvable, while another subset of problem instances with tracks of length six or more is demonstrated to be NP-complete.

Introduction

Rail transportation in the Netherlands is operated by the NS (*Netherlands Railways*) and more passengers are transported every day, which has led to an increase in required rolling stock. The vehicles are scheduled according to the timetable and when they are not in use, they are parked in shunting yards. The problem of parking and routing the trains in a shunting yard is known as the Train Unit Shunting Problem (TUSP), in which trains are considered as compositions of fixed-length train units (Freling et al. 2005). This is an NP-hard problem, and it remains a challenge to find good solutions to real-world scenarios in a reasonable time. Because of changes during the day, and as a sub-problem of finding a shunting plan, it is desirable to solve the scenarios in a few minutes or at most within half an hour. However, how to do this remains an open area of research. Moreover, current algorithms at the NS do not have a feasibility check implemented, thus resulting in long computation times for infeasible scenarios. Therefore, this paper considers a novel feasibility approach to the TUSP problem, to avoid these long computation times in practice.

Many shunting yards have a single-entry point and, with the use of switches, trains can be routed to different tracks in the yard. Such a shunting yard is regarded as a shuffleboard layout and can be abstractly represented by a tree (Huizingh 2018). Trees are recursive structures that do not include cycles, so there is one simple path from a node in the tree to its root, which forms the single-entry point. Yet, even on these relatively structured yards, no polynomial algorithms are known, and the complexity of the problem remains open.

One of the sub-problems of the TUSP is the routing component. A simplistic variant of this routing problem can be seen in the Pebble Motion (PM) problem, for which the feasibility of instances can be determined efficiently. Given a graph, a set of pebbles, and two configurations of these pebbles over the nodes of the graph, the question is whether there exists a sequence of moves to maneuver the pebbles from the initial to the goal configuration (Kornhauser, Miller, and Spirakis 1984). Each of these moves considers exactly one pebble that is moved from its current node to a neighboring node that was empty before this move operation. The main advantage of the PM problem is that it is solvable in linear time (Auletta et al. 1999). Therefore, insights from this problem could be useful in finding a better approach for studying the feasibility of the TUSP.

The main difference between the PM problem and the TUSP is that in the latter the pebbles are not yet present in the graph at the start of a scenario, and the PM problem can thus not be directly applied to TUSP instances. Therefore, this paper proposes a variant of the PM problem that takes the arrival and departure movements into account. Consider a shunting yard with a predefined input sequence of unique train units to be parked and a required output sequence of these train units. Is it possible to efficiently determine the feasibility of realizing the requested permutation between these sequences? This is similar to the real-world scenario where first all trains come in fixed order in at night after their shifts, and each train departs again in the morning, also

in a fixed order. This paper focuses on a process that is executed on several dead-end tracks of limited length, which are connected in a tree-like configuration, so the tracks can be seen as disjoint branches in the tree. The aim is to test the input-output permutation feasibility before trying to find a plan that enables this permutation.

Both the PM problem and the TUSP allow reallocation. In the TUSP context, this means that after trains are parked in the shunting yard, they can be moved intermediately to a different track, before departing the shunting yard (Wolfhagen 2017). Reallocation is often used to create way for another train which has to depart earlier. However, this process is also very costly. Shunting yards are often operated by a small set of drivers, who have to walk along the track to a train, move it, and then walk back. For safety of operations, only one train can move at a time. Moreover, in scenarios with tight planning and limited capacity, a reallocation movement can cause a delay, making the instance infeasible due to time restrictions. Therefore, this paper considers a PM variant without reallocation and focuses on the feasibility question raised only by the permutation between the input and output sequences. This is also related to a Multi-Agent Path Finding approach, which has been compared to the TUSP before (Mulderij et al. 2020).

In the proposed PM variant, a train unit is modeled by a pebble. Thus, train compositions can be modeled with consecutively scheduled pebbles in a sequence. Initially, train units are regarded to all have the same length, since the different train unit lengths do not vary too much in the real-world application. Afterward, an extension is studied that does consider train units with different lengths and more precise track lengths, to resemble real-world scenarios even more closely. In this variant, pebbles are given a size to indicate the length of the respective train unit. The PM variant with arrival and departure offers a novel approach to study the feasibility of the TUSP and reduce unnecessary computation times in practice, which is where this paper fills the current research gap.

The main contributions of this paper are:

- the introduction of a new arrival-departure PM variant including train compositions,
- a linear-time check for infeasibility of instances due to a limited number of tracks,
- a special case of the arrival-departure variant, with track lengths of two train units, that is polynomially solvable,
- a subset of the arrival-departure variant, with track lengths of six or more train units, that is NP-complete, and
- an extension of the arrival-departure variant with pebble sizes and parking track length that is NP-complete.

The next section gives related work on the TUSP and some history on the PM problem in relation to routing. Then, the problem is formally defined. Next, some terminology is given which is used in the feasibility approach discussed thereafter. Subsequently, the complexity of the proposed variant is examined, and finally, a problem extension with pebble sizes and track lengths is given.

Related Work

The Train Unit Shunting Problem has been widely studied from many angles. Di Stefano and Koči (2004) looked into a graph theoretical approach for train shunting at night. They distinguished different orders of shunting, based on which sides of the yard the trains can enter/depart. The Single Input Single Output track assignment is the most relevant to this paper. They conclude that the number of tracks necessary for parking n trains can be computed in $O(n \log n)$ time. Other work looking into finding this number of tracks used the chromatic number of the conflict graph, which is NP-hard (Føns 2006). Bohlin, Hansmann, and Zimmermann (2018) considered the complexity of finding the number of tracks for different scenarios, but not the Single Input Single Output type. Single Input was compared to Single and Double Output by a greedy solution algorithm approach proposed by Demange, Di Stefano, and Leroy-Beaulieu (2012). They considered a graph coloring angle for the online track assignment, though no other problem details were included. However, to the best of current knowledge, no study has focused on the feasibility of the TUSP.

A problem that is theoretically similar to TUSP is the Parallel Stack Loading Problem, which considers several parallel stacks to load and unload containers on. The loading and unloading can be compared to the arrival and departure of trains at a shunting yard, and the stacks are similar to Single Input Single Output tracks. Boge and Knust (2020) showed that this problem is NP-hard. This problem is related to the TUSP in a very abstract manner, as it considers the same approach of moving units through a space, which inspires new ways of looking at the problem.

The first notion of the Pebble Motion problem on graphs by Kornhauser, Miller, and Spirakis (1984) considered a tree of bi-connected components to offer a useful structure for moving pebbles. They presented a decision algorithm for instances with fewer pebbles than nodes in the graph, where the initial and goal configurations consist of the same nodes. Their approach divided the problem into transitive sub-problems, such that concluding one sub-problem is infeasible immediately infers the complete instance to be infeasible. This approach resulted in an efficient algorithm for determining the feasibility of the PM problem on a graph and emits plans with an upper bound of $\Theta(n^3)$ number of moves, where n is the number of pebbles.

Auletta et al. (1999) improved on this bound by providing a linear-time algorithm for determining the feasibility of the Pebble Motion problem on Trees. Their algorithm resulted in plans of length $O(n^2(N-n))$ moves, where n is the number of pebbles and N is the number of nodes. This is a fairly high upper bound and has been reviewed in the literature as much higher than the number of moves needed for real-world instances (Goraly and Hassin 2010). For train shunting in particular, the number of moves needs to be very small, as each move requires a driver to move the train manually and walk across the shunting yard. Therefore, this result is not directly applicable to the PM variant proposed by this paper.

Related to the PM problem is the Multi-Agent Path Finding problem, where multiple agents have to be routed

through a graph from their start to their goal location (Surynek 2009b), and the similar Multi-robot Path Planning problem, where the robots are controlled as a single entity (Surynek 2009a). However, most of the research in this direction has focused on path planning and finding (sub)optimal solutions to these problems (Krontiris, Luna, and Bekris 2013). On the contrary, work on the PM problem is more focused on efficiently answering the feasibility question. One application of the PM problem involved 3D movements through a space (Krontiris et al. 2014), where a Manipulation Pebble Graph is constructed between poses to check whether the rearrangement is possible. Here, the PM problem is used for this feasibility check, although no size of the pebbles is taken into account.

The Multi-Agent Path Finding problem has been related to the TUSP in an attempt to gain insights into solving the latter (Mulderij et al. 2020). However, the additional details that were added to the Multi-Agent Path Finding problem extension led to an increase in complexity, and the problem was studied from an optimality perspective, instead of a feasibility one. To the best of current knowledge, no work has been done on the relation between the PM problem and the TUSP, nor has the former been considered with the arrival and departure of pebbles. Furthermore, no extensions with pebble sizes currently exist.

Problem Definition

This section starts with the original Pebble Motion on Trees problem, then gives the variant of this considered in this paper, which includes the arrival and departure of pebbles.

This paper focuses on the shuffleboard shunting yards, and thereby regards the underlying graph as a tree, where the trains enter and depart at the root node. In this paper, each pebble represents a unique train unit. Therefore, the new variant is based on the Pebble Motion on Trees problem, which is formally defined below (Auletta et al. 1999). This is a well-known case of the general Pebble Motion problem that only considers instances where the graph is a tree. A configuration C assigns each of the pebbles in P a node in T such that no two pebbles occupy the same node.

Problem: Pebble Motion on Trees problem

Input: $I = (T, P, C_A, C_D)$: given is a tree $T = (V, E)$; a set P of $n < |V|$ pebbles;an arriving configuration $C_A \subseteq P \times V$ of these pebbles; and a departing configuration $C_D \subseteq P \times V$.

Question: Is there a sequence of moves, which each transfers a pebble from its current position $v \in V$ to an adjacent unoccupied node $v' \in V$ s.t. $\{v, v'\} \in E$, to move the pebbles from C_A to C_D?

First, a variant of this problem will be created that includes arrival and departure of pebbles. Say there are n pebbles, which arrive in a sequence $S = (s_1, .., s_n)$ and depart in a sequence $D = (d_1, .., d_n)$. The arriving sequence is defined as the first to the last arrival, while the departing sequence is defined as the last departure to the first departure. This ensures that when $S = D$, the order will be completely Last In First Out, so the solution is straightforward: the instance is feasible if there is sufficient capacity.

The instances considered in this paper assume that the last arrival is always scheduled before the first arrival, which is the real-world scenario of a shunting yard at night. Moreover, no reallocation is allowed, resembling real-world scenarios as discussed in the introduction. Finally, in this paper, pebbles are considered to be unique, and pebbles must be uniquely matched between the sequences S and D. In other words, when types or colors are considered, as also done in PM literature (Goraly and Hassin 2010), there are n pebbles and n different colors which are perfectly matched to the pebbles.

To model the arrivals and departures, a simple path L of length n (the number of pebbles) is added to the tree that can be used to position the pebbles in the correct sequence to form the arriving and departing sequences configurations (see Definition 1). This paper considers the following (novel) variant of the Pebble Motion problem, the PMTAD problem.

Problem: Pebble Motion problem on a Tree with Arrival and Departure (PMTAD)

Input: $I = (P, S, D, T_L)$: a set P of n unique pebbles with a size of exactly one node each; an arriving sequence S of the pebbles; a departing sequence D of the pebbles; and a *shunting tree* T_L, which is the tree representation T of a shunting yard, extended at the root node with a path L of length n, which is used for positioning the arriving and departing configurations.

Question: Is there a sequence of moves that first transfer all pebbles from their arriving position on L to a parking location in the tree T, and then transfer all pebbles from their parking location in T back to their departing position on L, without intermediate moves?

Here, all pebbles are unique and represent train units which are all the same size (the number of carriages). The sizes of pebbles are later introduced in the Problem Extension with Pebble Size and Track Length. Train compositions can still be formed by putting two (or more) train units that belong to the same composition after each other in the arriving sequence. They can be split up or not, depending on the departing sequence. The *shunting tree* as used in this problem variant is illustrated in Figure 1. The nodes on the path L can thus not be used for parking pebbles.

Definition 1 (Shunting tree)**.** *A **shunting tree** $T_L = \{T \cup L \cup e_T\}$ consists of the tree T, rooted by node t_0, which is connected to the path $L = (v_1, .., v_n)$ through the edge $e_T = \{t_0, v_1\}$, such that removing this edge e_T would disconnect T from L.*

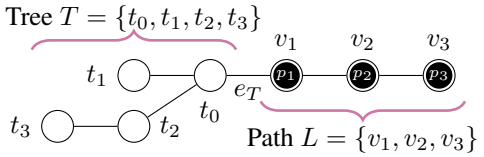


Figure 1: Definition of a shunting tree.

Partitioning the Pebbles

This section introduces terminology which is used in the hereafter discussed feasibility approach.

Some patterns in a sequence can easily be determined from the arrival and departure order, so the solution can be quickly determined. Say there are three pebbles (p_1, p_2, p_3) that arrive in that order, and the departure sequence is the same; meaning that first p_3 departs, then p_2, and finally p_1. In this case, the pebbles can park in a tree with sufficient space and depart when necessary, without intermediate re-ordering movements. The same principle can be applied to pebbles that do not arrive exactly after each other, but are still in the same relative order. For example, if the arriving sequence is (p_1, p_3, p_2, p_4) and they depart as (p_1, p_2, p_3, p_4), then p_1 and p_2 are still in the same relative order, and so are p_3 and p_4, so both pairs could be parked together such that they can depart in their correct order, as long as the two pairs are not parked together as well (see Figure 2c).

Now, this idea is employed to determine the ordered sets of pebbles that are in the correct relative order, which is called a partition $\Pi(S)$ of the arriving sequence, compared to the departing sequence D. Here, the latter is assumed to always be the ordered sequence $(p_1, p_2, \ldots, p_n)$. A partition is a structure that is often used to signal parts of the problem that belong together, and is more frequently used in newly identified relations (Lindner and Liebchen 2019). Since the order of the pebbles is considered, each set of the partition is referred to as a **totally ordered set (toset)** because between each pair of pebbles within such a set their order in S and D is respected. The partition is defined in Definition 2, and there can be different partitions for a sequence S. Consider, again, the previous example of arrival sequence (p_1, p_3, p_2, p_4) and departure sequence (p_1, p_2, p_3, p_4). One way of defining the partition is $\Pi_1(S) = \{(p_1, p_2), (p_3, p_4)\}$, but $\Pi_2(S) = \{(p_1, p_3), (p_2, p_4)\}$ is also a valid partition. This example is shown in Figure 2.

Definition 2 (Partition)**.** *A **partition** $\Pi(S)$ of an arriving sequence S is the set of tosets $\Pi(S) = \{\pi_1, .., \pi_m\}$ such that*

- **i)** *all tosets are disjoint: $\forall \pi_j \in \Pi(S)$ s.t. $\pi_i \neq \pi_j : \pi_i \cap \pi_j = \emptyset$,*
- **ii)** *the union of the tosets includes all pebbles: $\bigcup \pi_i \in \Pi(S) = P$, thus $\sum_{\pi_i \in \Pi(S)} |\pi_i| = n$, and*
- **iii)** *the tosets respect the orders of S and D: $S(p_k) < S(p_l) \wedge D(p_k) < D(p_l)\,, 1 \leq k < l \leq |\pi_i|, \forall \pi_i \in \Pi(S)$, in other words, two pebbles in the same toset appear in the same order in the toset as they do in S and D.*

The partition of a sequence can be used to determine which pebbles can be parked together. The problem is based on a shunting tree T, which is given in practice, and the tree T can be expressed in a set of disjoint branches $B(T)$. The branches have to be disjoint so that they are always reachable from the root node. Otherwise, consider a subtree shaped like T in Figure 1, such that t_0 is not the root. If the branches are $\{(t_0, t_1), (t_0, t_2, t_3)\}$ and a toset of size two is parked on branch (t_0, t_1), then the pebble parked on t_0 blocks other pebbles from reaching the second branch. Therefore, branches must always be disjoint and start at the child of a branching node (t_0 in this case). The branch set is defined in Definition 3.

Definition 3 (Branch set)**.** *A **branch set** $B(T)$ of the tree $T = (V, E)$ is the set $B(T) = \{b_i\}$, where each branch $b_i \subset V$ is a set of nodes such that:*

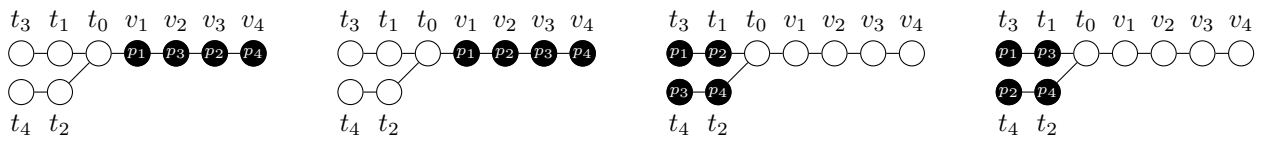
- **i)** *all branches are disjoint: $\forall b_i, b_j \in B(T)$ s.t. $b_i \neq b_j : b_i \cap b_j = \emptyset$,*
- **ii)** *a branch begins at the child of a* branching node *$v \in V$, such that v has more than one child node,*
- **iii)** *a branch is a subtree of T, and*
- **iv)** *the length ℓ_i of a branch b_i is given by the number of nodes it includes.*

Each branch b can be used to park the pebbles associated with a single toset π if the branch is large enough to hold $|\pi|$ pebbles. However, the branch set is not unique to a tree, unless each branch is a simple path. Otherwise, the tree can be one very large branch that includes subbranches, or each branch can be defined individually.

To determine the feasibility of a given instance of the PMTAD problem with a sequence S and a tree T, a pairwise comparison of the partition $\Pi(S)$ and branch set $B(T)$ is used. The relation between a partition and branch set is expressed by whether they are pairwise comparable (Corollary 1).

Corollary 1 (Pairwise comparable)**.** *Since branches are disjoint, just like the tosets in a partition, it follows that these can mapped one-to-one to provide a sufficient condition for feasibility. This is referred to as the **pairwise comparability** of the branch set and partition.*

As both the partition of a sequence and the branch set of a tree are neither uniquely defined, it is not a trivial problem to determine whether they are pairwise comparable, and a scenario is thus feasible. For example, consider an instance of the PMTAD problem where T has two branches with each two nodes (like in Figure 2), $P = \{p_1, p_2, p_3, p_4\}$, $S = (p_1, p_3, p_2, p_4)$, and $D = (p_1, p_2, p_3, p_4)$. The configuration



(a) $S = (p_1, p_3, p_2, p_4)$ (b) $D = (p_1, p_2, p_3, p_4)$ (c) $\Pi_1(S) = \{(p_1, p_2), (p_3, p_4)\}$ (d) $\Pi_2(S) = \{(p_1, p_3), (p_2, p_4)\}$

Figure 2: Different partitions for arriving sequence $S = (p_1, p_3, p_2, p_4)$ and departing sequence $D = (p_1, p_2, p_3, p_4)$.

of pebbles like in Figure 2d clearly shows this instance is feasible, because the partition $\Pi_2(S) = \{(p_1, p_3), (p_2, p_4)\}$ matches the branch set. However, the also valid partition $\Pi_3(S) = \{(p_1, p_3, p_4), (p_2)\}$ does not match the branch set in this tree because each toset must be assigned a subset of branches that is disjoint with the other branches. So, the partition Π_2 immediately shows the scenario is feasible, while the validity of the partition does not imply the feasibility of a scenario, like for partition Π_3.

Therefore, the next problem (PPST) takes a given branch set of a shunting tree and the question is whether a partition exists that is pairwise comparable with that branch set.

Problem: Partition for a Pebble Sequence on a Tree (PPST)

Input: $I = (B, P, S, D)$: given is a branch set $B(T)$ based on a shunting tree T_L; a set P of n unique pebbles; an arriving sequence of pebbles S; and a departing sequence over the pebbles D.

Question: Is there a valid *partition* $\Pi(S)$ into tosets $\Pi_1, .., \Pi_m$ such that the tosets of $\Pi(S)$ and the individual branches of $B(T)$ are *pairwise comparable*?

An answer of the PPST, in the form of a partition, means that there exists a parking configuration of pebbles such that the associated PMTAD instance is also feasible (see proof in supplement Section 4[1]). However, it can be very difficult to determine whether a partition exists. Consider, for example, the scenario in Figure 3, is the permutation between S and D possible given this tree? The next section gives an approach to decide this example.

Feasibility Approach

This section includes several Lemmas that can be used to decide the (in)feasibility of a PPST instance. At the end, these Lemmas are combined into an efficient feasibility approach, of which some elementary results are given.

To help determine the feasibility of an instance, and thus check the compatibility of the branch set and partition, a Directed Acyclic Graph (DAG) of the sequence can be constructed, as described in Method 1. Here, the $DAG(S)^+$ and $DAG(S)^-$ are introduced, where the former gives the relations between pebbles that are compatible and can be in the same toset, while the latter indicates the pebbles that cannot be in the same toset. By displaying the nodes of these graphs on an $n \times n$ grid, the two different DAGs can be clearly distinguished, and the coefficients of the connections are obvious. However, this grid is not necessary for the validity of the lemmas introduced hereafter.

Existing graph algorithms can be executed on the constructed DAGs to determine the feasibility of certain instances. Take the Longest Path problem (definition in supplement Section 5) on the $DAG(S)^-$, in which pebbles of respectively connected nodes cannot be parked together. Therefore, the length of the longest path of these pebbles determines the minimal number of disjoint branches that are necessary for a feasible solution for the PPST problem. As an example, the DAG in Figure 4 matches the sequence in Figure 3, and we can thus conclude the length of the longest path in the $DAG(S)^-$ (three) is higher than the number of branches (two), so this example is infeasible.

Since the longest path can be found in $O(n)$ time in a DAG (Sedgewick 2011), this is an improvement to the $O(n \log n)$ bound found by Di Stefano and Koči (2004). This is a great speed-up and can also be used as an infeasibility check: when there are fewer branches than the length of the longest path in the $DAG(S)^-$, the instance is infeasible (Lemma 1, proof in supplement Section 6). Although it should be noted that there are other checks for infeasibility, so this is not a necessary condition.

Lemma 1 (Minimal number of branches)**.** *If there are fewer branches than the length of the longest path $L(S)^-$ in the $DAG(S)^-$, i.e. $|B(T)| < |L(S)^-|$, then I is infeasible.*

On the other hand, the longest path in the $DAG(S)^+$ represents the maximum number of pebbles that can be in a toset together. If there exists a branch of that length in the tree, then this branch can be filled with pebbles from that toset. However, if the largest branch in $B(T)$ has a length k and the longest path in the $DAG(S)^+$ has a length $l < k$, the

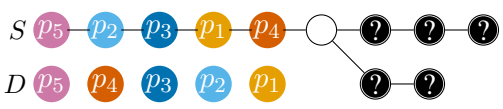


Figure 3: Is this instance feasible?

[1]Visit the supplementary material at *rebrand.ly/supmat*.

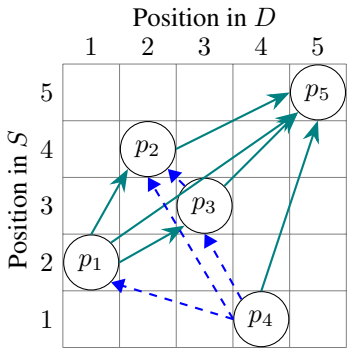


Figure 4: Directed acyclic graph of $S = (p_4, p_1, p_3, p_2, p_5)$.

Method 1 (Constructing a DAG of a sequence)**.** *Given pebble sequences S and D of length n:*

1. Construct an $n \times n$ grid with the departure positions on the x-axis and the arrival positions on the y-axis.
2. Create a point for each pebble p: the row is the position of p in S and the column is the position of p in D.
3. Create the positive graph $DAG(S)^+$ by connecting the point of pebble p with *green* to every point of a pebble that arrives earlier but departs later (these are the lines with a positive coefficient).
4. Create the negative graph $DAG(S)^-$ by connecting the point of pebble p with dashed *blue* to every point of a pebble that arrives earlier and departs earlier (these are the lines with negative coefficient).

instance is only feasible if there are more than $k - l$ empty nodes in the graph. This is most easily visualized when there are exactly as many nodes in the tree as there are pebbles. If the largest branch cannot be filled with pebbles from one toset then there will be pebbles that cannot be parked, so the instance is infeasible. Take $c \geq 0$ to be the difference between n and the total number of nodes in the branch set (Lemma 2, see proof in supplement Section 7).

As an example, if a tree would have one branch of length four and one branch of length one, then $c = 0$, because there are five nodes in $B(T)$ and five pebbles. The longest path of length three in the $DAG(S)^+$, e.g. (p_1, p_2, p_5). The branch of length four cannot be filled, so it is infeasible. However, given the largest branch of length three (as in Figure 3), these pebbles can be parked, though the scenario is still infeasible (Lemma 1), so Lemma 2 is not a necessary condition for feasibility.

Lemma 2 (Parking in the largest branch)**.** *For $c \geq 0$ empty nodes, if $\sum_{b \in B} |b| = n + c$ and $|B(T)| > |L(S)^+| + c$, then the instance is infeasible.*

Besides this infeasibility condition, the $DAG(S)^+$ can also be used to find a feasible solution. Since the pebbles that are connected in the $DAG(S)^+$ are in the correct relative order, they can be together in a toset of the partition. Therefore, a path cover (definition in supplement Section 5) of the $DAG(S)^+$ gives a possible partition of the sequence (Lemma 3, see proof in supplement Section 8). Moreover, given the number of branches $m = |B(T)|$, a path cover with m paths can be found in polynomial time. However, the lengths of the individual branches cannot be included in this search, unless all exponentially many options are exhausted, so it is not a complete solution to the PPST problem.

As an example, a path cover of the $DAG(S)^+$ gives the partition $\{(p_1, p_2, p_5), (p_3), (p_4)\}$. If the tree in Figure 3 would two branches of length one instead of the branch of length two, then this partition would fit, and the instance would be feasible.

Lemma 3 (Find a vertex-disjoint path cover)**.** *Given the number of branches $|B(T)|$, a vertex-disjoint path cover of size $K = |B(T)|$ can be found.*

The three lemmas introduced here can be used to create an initial proof-of-concept algorithm to determine the feasibility of an instance given sequence S and branch set B. First, Lemma 1 can be used to check if there are enough branches in B. If not, return INFEASIBLE. Then, take the longest path from the $DAG(S)^+$ (or a random one if there are several of the same length) that fits in the largest branch, and fill branches in non-increasing order. If all pebbles are parked, return FEASIBLE, else return UNKNOWN because the three Lemmas cannot provide a decision.

In Figure 5, the results are shown for all possible $4! = 24$ sequences of length four, on four different branch sets $B \in \{(4), (3, 1)(2, 2), (2, 1, 1)\}$, since the branch set $(1, 1, 1, 1)$ is always feasible for a sequence of length four. For longer sequences, the number of UNKNOWN returns is expected to go up, because cases like the previous example will occur more often. Each instance was tested 10 times and the averages are shown in the figure. Between the 10 runs, the input

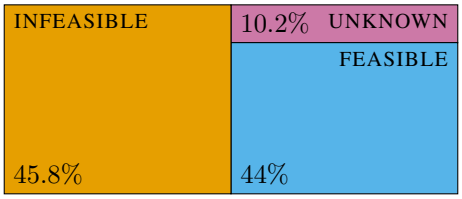


Figure 5: Results for length-4 sequences on four branch sets.

did not change, but due to the random drawing of the longest path in the $DAG(S)^+$, some feasible instances were sometimes marked UNKNOWN. For example, $S = (p_1, p_4, p_2, p_3)$ is feasible on branch set $(2, 2)$, but only if (p_1, p_4) is a toset, and if path (p_1, p_3) is selected first, the feasible solution will not be found and UNKNOWN is returned.

Complexity of the Problem

The approach discussed so far cannot determine the feasibility of the PPST problem in polynomial time for all instances. So, the complexity of the overall problem is studied, and two subsets of problem instances are distinguished in this section.

Here, ℓ_{max} is used to say that all branches have the same length of ℓ_{max}. In the first case, there are $m = |B(T)|$ branches, each of length $\ell_{max} = 2$, and $n \leq 2m$ pebbles. This 2-PPST variant can be solved in polynomial time (Theorem 1, proof in supplement Section 2), which can be shown by transforming the problem to a bipartite matching problem, to which the solution can be found in polynomial time (Sedgewick 2004). The idea is that a bipartite matching will create pairs of pebbles that are compatible for the same toset and can thus be parked on the same branch. A bipartite graph can be constructed by creating a node $v_p^\flat$ and $v_p^\sharp$ for each pebble p and an edge $\{u_p^\flat, v_q^\sharp\}$ exists in the bipartite graph if pebble p arrives earlier than pebble q but departs later.

Theorem 1 (2-PPST is in P)**.** *The Partition for a Pebble Sequence on a Tree problem is in P if there are $m = |B(T)|$ branches of length $\ell_{max} = 2$ and $n \leq 2m$ pebbles.*

The other subset concerns instances where the length of the branches is fixed to six (or more) nodes. Then, a reduction can be constructed from the known Mutual Exclusion Scheduling problem (MES). The MES problem takes a set of jobs that have to be scheduled on a set of machines within a given number of time slots. Furthermore, a relation between the jobs is given that indicates which jobs cannot be processed simultaneously. The MES problem can also be thought of in graphical terms: given an undirected graph $G = (V, E)$ with a node for each job and an edge between every pair of conflicting jobs; then any subset $V' \subset V$ of jobs, for which $|V'| \leq M$ and V' is an independent set of G, can be executed at each time step (Jansen 2003).

The idea behind the reduction is that these independent sets of jobs can be thought of as tosets, since these jobs cannot be conflicting, just like pebbles in tosets cannot be conflicting. Thus, the jobs that are processed at each time step correspond to the pebbles that are parked on the same

branch. The conflicts between jobs can be used to define the sequences, which becomes apparent with the introduction of the permutation graph.

A specific variant of the MES problem uses a permutation graph for these relations, and this variant has been shown to be NP-hard for a fixed constant $M \geq 6$ number of machines (Jansen 2003). A permutation $\Psi = (\psi_1, .., \psi_N)$ of the jobs $\{1, .., N\}$ is given along with the input. $\Psi^{-1}(i)$ gives the position of job i in the permutation. The **permutation graph** $G_\Psi = (V, E_\Psi)$ with $V = \{1, .., N\}$ has an edge $\{i, j\} \in E$ if and only if $i < j$ and $\psi_i > \psi_j$ (Boge and Knust 2020). The MES problem on a permutation graph is introduced below.

Problem: Mutual Exclusion Scheduling problem on a permutation graph (MESP)

Input: $I = (V, U, M, \Psi)$: given is a set V of N jobs, a number of time slots U, a number of machines M, and a permutation graph $\Psi = (V, E_\Psi)$;

Question: Is there a partition of V into at most U independent sets $W_u : u = 1, .., U$ (independent means that there is no edge $\{i, j\} \in E_\Psi$ for any pair $i, j \in W_u$) with $|W_u| \leq M$ for all u?

Since the arrival sequence of pebbles is a permutation of the departure sequence, this has a clear relation to the permutation graph that shows the conflicting jobs. Jobs that conflict with each other have a different order in the permutation than in the original sequence, same as the pebbles which conflict between the arrival and departure sequences.

A simple example to illustrate this: instance I of the MESP has M = 4 machines, U = 3 time slots, N = 10 jobs V = $\{1, .., 10\}$, and the permutation $\Psi = (4, 1, 3, 10, 2, 6, 5, 8, 7, 9)$. The constructed instance I' of the PPST has n = N = 10 pebbles with $S = (p_4, p_1, p_3, p_{10}, p_2, p_6, p_5, p_8, p_7, p_9)$ and $D = (p_1, .., p_{10})$, where the branch set B consists of m = U = 3 branches each of length ℓ_{max} = M = 4 nodes. For example, a feasible solution for I consists of the three independent sets $W_1 = \{1, 7, 9, 10\}$ which corresponds to pebble toset $\pi_1 = (p_4, p_5, p_7, p_9)$; $W_2 = \{5, 6, 8\}$ with toset $\pi_2 = (p_2, p_6, p_8)$; and $W_3 = \{2, 3, 4\}$ with toset $\pi_3 = (p_1, p_3, p_{10})$. These tosets can form a partition, such that there are exactly $U = 3$ branches used and on each branch at most $M = 4$ pebbles can be parked that have a correct order in both S and D.

Theorem 2 (PPST is NP-complete)**.** *The Partition for a Pebble Sequence on a Tree problem is NP-complete if all branches have length $\ell_{max} \geq 6$ nodes.*

Proof. PPST $\in$ NP: Given a solution $\Pi(S)$ to an instance $I' = (B, P, S, D)$ of the PPST problem, it can be established in $O(n)$ time whether the solution is a valid solution of the PPST by checking all conditions of a valid partition and the pairwise comparability.

PPST $\in$ NP-HARD: Given an instance $I = (V, U, M, \Psi)$ of the MESP problem, construct an instance $I' = (B, P, S, D)$ as follows. For each job $i \in V$, create a pebble $p_i \in P$. Set $S = \Psi$ and $D = (p_1, .., p_n)$, such that a pebble p_i has position $\Psi^{-1}(i)$ in the arriving sequence. Moreover, create $m = U$ branches in B, with each a length of $\ell_{max} = M$ nodes. All these steps can be performed in $O(n)$ time, so the reduction is polynomial. Next, it is shown that I is a yes-instance of the MESP if ($\Longrightarrow$) and only if ($\Longleftarrow$) I' is a yes-instance of the PPST.

PROOF OF ($\Longrightarrow$): Suppose I is a yes-instance with at most U independent sets W_u, each containing at most M jobs. Two arbitrarily chosen jobs $i > j$ of an independent set W_u must satisfy $\psi_i > \psi_j$, since otherwise there would be an edge in the conflict graph. Thus, all items of one independent set can be put into one toset, such that all U tosets are disjoint, all pebbles are in exactly one toset (because all jobs must be scheduled on one machine) and the tosets respect the orders of S and D. Therefore, a valid *partition* is possible. Furthermore, due to $W_u \leq M$ for all u, there is no toset with more pebbles than the branch length M and there are no more than U branches used such that the pebbles of each toset can be parked together on a branch. So, the valid partition is *pairwise comparable* with the branches and thus I' is a yes-instance of the PPST.

PROOF OF ($\Longleftarrow$): Suppose I' is a yes-instance of the PPST. Then, there exists a partition Π that is pairwise comparable with branch set B. Each of the tosets contains at most M elements by construction, because of the branch length. Furthermore, each toset is parked on a single branch, so there are no more than U branches. Since the tosets satisfy the orders of S and D, for any two pebbles p_i, p_j in a toset, where $i < j$, it must hold that $\Psi^{-1}(p_i) < \Psi^{-1}(p_j)$. This implies there cannot be an edge in the permutation graph. So, all pebbles of one toset form an independent set consisting of at most M and all tosets together form a partition into at most U independent sets. □

To conclude, there is no complete approach to solve the PPST, and thus the PMTAD, problem. A proof-of-concept algorithm was shown that can determine about 90% of the (small) instances. Furthermore, for all instances of 2-PPST the problem is in P, although for all instances where $\ell_{max} \geq 6$, the problem is NP-hard.

Problem Extension with Pebble Size and Track Length

The variant in the previous sections regarded single train units and looked more into the feasibility regarding the number of train units. However, there are slight differences in the number of carriages per train unit, so this can influence the feasibility of an instance. Therefore, an extension of the PMTAD problem is proposed that covers this difference. A pebble still represents one train unit, but now also has a size attribute determined by the number of carriages. Similarly, a branch now needs to fit both the number of train units, and the total number of carriages that those train units consist of. The branch length is defined on the edge of a branch, which connects a branching node (a node with two or more children) to the branch. The length and sizes could also be expressed in meters, but since train carriages are always the same length (at NS), this paper sticks to carriages for simplicity. This problem is defined as the PMTADL problem.

Problem: Pebble Motion problem on a Tree with Arrival, Departure, and Length inclusion (PMTADL)

Input: $I = (P, S, D, T_L, E_B, \ell, \lambda)$: a set P of n unique pebbles with a size of exactly one node each; an arriving sequence S of the pebbles; a departing sequence D of the pebbles; and a *shunting tree* T_L, which is the tree representation T of a shunting yard, extended at the root node with a path L of length n, which is used for positioning the arriving and departing configurations; a set of branch edges E_B which connect a branch to its branching node; a function $\ell(e)$ which returns the length of an edge e; and a function $\lambda(p)$ which returns the size of a pebble p.

Question: Is there a sequence of moves that first transfers all pebbles from their arriving position on L to a parking location in the tree T, and then transfers all pebbles from their parking location in T back to their departing position on L, without intermediate moves, such that the edge capacity $\ell(e)$ for all $e \in E_B$ is not exceeded by the sum of the sizes of the pebbles parked in the branch of this branching edge e?

The original PMTAD problem is a special case of the PMTADL problem, where $\lambda(p) = 1, \forall p \in P$ and $\ell(e_t) = 1, \forall t \in T$ and the length of a branch $b \subset B(T)$ was simply the number of nodes in the branch. So, to use similar notation as before, pairwise comparability in length is introduced (Corollary 2).

Corollary 2 (Pairwise comparable in length)**.** *For a partition and branch set to be* ***lengthwise pairwise comparable****, building on Corollary 1, the branch capacity must be also respected, i.e.* $\sum_{p \in \pi_i} \lambda(p) \leq \ell(e_{b_i}), 1 \leq i \leq |\Pi(S)|$.

Next, an extension of the PPST, the PPSTL, is proposed that can solve the PMTADL problem like the PPST solves the PMTAD problem (supplement Section 4).

Problem: Partition for a Pebble Sequence on a Tree with Length inclusion (PPSTL)

Input: $I = (B, P, S, D, \ell, \lambda)$: given is a branch set $B(T)$ based on a shunting tree T_L; a set P of n unique pebbles; an arriving sequence of pebbles S; a departing sequence of pebbles D; an edge length function ℓ; and a pebble size function λ.

Question: Is there a valid *partition* $\Pi(S)$ into tosets $\Pi_1, .., \Pi_m$ such that the tosets of $\Pi(S)$ and the individual branches of $B(T)$ are *lengthwise pairwise comparable*?

To create a reduction showing the PPSTL is NP-complete (Theorem 3), the NP-hard Partition Problem (PP) is introduced (Garey and Johnson 1979). Note, that there can only be a solution to the PP if the $\sum(X)$ is even, otherwise, no two subsets can be the same size.

Problem: Partition problem (PP)

Input: $I = (X)$: given is the multi-set X of positive integers.

Question: Is there a partition of X into two disjoint subsets X_1 and X_2 such that the sum of the numbers in X_1 equals the sum of the numbers in X_2?

The intuition of the reduction is as follows. Since each element of the multi-set in the Partition problem can be seen as an item with a weight equal to its integer value, these integers can be associated with the sizes of pebbles. Furthermore, since there is only limited space in each of the two subsets, this is similar to $m = 2$ branches with a certain length. Then, the only remaining attributes of the PPSTL problem to be defined are the pebble sequences, and when these are equal ($S = D$), the main problem is to find out if there is enough space in the two branches, similar as in the classic Partition Problem. This reduction leads to the result of Theorem 3 and its formal proof is given in supplement Section 3.

Theorem 3 (PPSTL is NP-complete)**.** *The Partition for a Pebble Sequence on a Tree with Length inclusion problem is NP-complete.*

Since the PPSTL is an NP-complete problem, it is impossible to determine whether a scenario is feasible in polynomial time (unless P = NP).

Conclusion

To avoid long computation times on infeasible instances, this paper studies the feasibility of the Train Unit Shunting Problem. A variant of the Pebble Motion on Trees problem is proposed that includes the arrival and departure of pebbles in the tree: the PMTAD problem, where no reallocation is allowed. This variant is representative of the TUSP because the PMTAD problem includes sequences of train units that must be parked or permuted in a limited space. A related problem (PPST) is defined that, given a branch set of a shunting tree, finds a partition of a pebble sequence, which is shown to be a good measure for feasibility. A solution approach is given that can decide the feasibility on the majority of (small) instances, using derived conditions on the number of branches and their length. Furthermore, a subset of problem instances with m branches of length $\ell_{max} = 2$ and $n \leq 2m$ pebbles can be solved in polynomial time. Finally, there is a subset of the problem instances, with a fixed branch length $\ell_{max} \geq 6$, that is NP-complete. Another contribution is the PMTAD extension with pebble sizes, the PMTADL problem, and its related PPSTL problem, which is shown to be NP-complete. The relations between the different problem variants discussed are shown in supplement Section 1.

This paper only considers scenarios where the underlying graph can be represented by a tree, based on the common shuffleboard-layout shunting yard, though future research could extend this work to general graphs. The remaining gap for branch lengths $\ell_{max} = 3, 4, 5$ remains an open question but might be applicable for a bounded search tree algorithm, such that the given lemmas can limit the size of the problem instance, thereby reducing the overall computation time. Finally, this paper assumes no reallocation is possible as reallocation is an expensive process that requires a driver to walk over to the train, move the train, and walk back. Moreover, in limited capacity shunting yards with tight planning, reallocation can cause delays. A theory is provided that mostly applies to parking situations, yet the addition of reallocation could lead to new insights and more application possibilities. The definitions and results in this paper provide a framework for studying such other variants.

References

Auletta, V.; Monti, A.; Parente, M.; and Persiano, P. 1999. A Linear-Time Algorithm for the Feasibility of Pebble Motion on Trees. *Algorithmica*, 23: 223–245.

Boge, S.; and Knust, S. 2020. The parallel stack loading problem minimizing the number of reshuffles in the retrieval stage. *European Journal of Operational Research*, 280(3): 940–952.

Bohlin, M.; Hansmann, R.; and Zimmermann, U. T. 2018. *Optimization of Railway Freight Shunting*, 181–212. Cham: Springer International Publishing. ISBN 978-3-319-72153-8.

Demange, M.; Di Stefano, G.; and Leroy-Beaulieu, B. 2012. On the online track assignment problem. *Discrete Applied Mathematics*, 160(7-8): 1072–1093.

Di Stefano, G.; and Koči, M. L. 2004. A graph theoretical approach to the shunting problem. *Electronic Notes in Theoretical Computer Science*, 92: 16–33.

Freling, R.; Lentink, R. M.; Kroon, L. G.; and Huisman, D. 2005. Shunting of passenger train units in a railway station. *Transportation Science*, 39(2): 261–272.

Føns, P. 2006. *Decision Support for Depot Planning in the Railway Industry*. MSc Thesis, Technical University of Denmark.

Garey, M. R.; and Johnson, D. S. 1979. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. United States of America: Bell Telephone Laboratories, Incorporated. ISBN 0-7167-1044-7.

Goraly, G.; and Hassin, R. 2010. Multi-color pebble motion on graphs. *Algorithmica*, 58(3): 610–636.

Huizingh, E. 2018. *Planning first-line services on NS service stations*. MSc Thesis, University of Twente, Enschede.

Jansen, K. 2003. The mutual exclusion scheduling problem for permutation and comparability graphs. *Information and Computation*, 180(2): 71–81.

Kornhauser, D. M.; Miller, G.; and Spirakis, P. 1984. *Coordinating pebble motion on graphs, the diameter of permutation groups, and applications*. MSc Thesis, M. I. T., Dept. of Electrical Engineering and Computer Science.

Krontiris, A.; Luna, R.; and Bekris, K. E. 2013. From feasibility tests to path planners for multi-agent pathfinding. In *Sixth annual symposium on combinatorial search*.

Krontiris, A.; Shome, R.; Dobson, A.; Kimmel, A.; and Bekris, K. 2014. Rearranging similar objects with a manipulator using pebble graphs. In *2014 IEEE-RAS International Conference on Humanoid Robots*, 1081–1087.

Lindner, N.; and Liebchen, C. 2019. New perspectives on PESP: T-partitions and separators. In *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Mulderij, J.; Huisman, B.; Tönissen, D.; van der Linden, K.; and de Weerdt, M. 2020. Train Unit Shunting and Servicing: a Real-Life Application of Multi-Agent Path Finding. *arXiv preprint arXiv:2006.10422*.

Sedgewick, K. D., Robert; Wayne. 2011. *Algorithms*. Addison-Wesley Professional, 4th edition.

Sedgewick, R. 2004. *Algorithms in Java, Part 5 graph algorithm*. Addison-Wesley Pearson Education, 3rd edition.

Surynek, P. 2009a. An application of pebble motion on graphs to abstract multi-robot path planning. In *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, 151–158. IEEE.

Surynek, P. 2009b. A novel approach to path planning for multiple robots in bi-connected graphs. In *2009 IEEE International Conference on Robotics and Automation*, 3613–3619. IEEE.

Wolfhagen, F. 2017. *The train unit shunting problem with reallocation*. MSc Thesis, Erasmus University Rotterdam.