Comparing planners for rail planning in PDDL

How multiple shunting yards can be created in PDDL to replicate real-world scenarios

Tim Tian[1]

Supervisor(s): Sebastijan Dumančić[1]**, Issa Hanou**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Tim Tian
Final project course: CSE3000 Research Project
Thesis committee: Sebastijan Dumančić, Issa Hanou, Rihan Hai

Abstract

This paper explored the usage of the Planning Domain Definition Language in the Train Unit Shunting Problem or TUSP, an NP-hard problem that occurs in train storage. This paper focuses on the evaluation and improvement of existing planners to solve TUSP with multiple shunting yard layouts, as well as an appropriate domain with it. Starting with the implementation of multiple shunting yards in the domain, by connecting train tracks in problem instances. Followed by the evaluation of the planners, where planners are evaluated on speed, plan cost and solvability. Among the evaluated planners, the *Team4* planner from the International Planning Competition 2018 demonstrates exceptional performance by successfully solving all problem instances and having the highest speed overall. Finally, the *Team4* planner was optimised, specifically by improving the domain for Last In First Out, LIFO, tracks, since it encountered difficulties when solving problems that contained these tracks. The tracks were given new predicates and actions, resulting in much higher speeds for problems that consisted of LIFO tracks. Results suggest that a suitable planner has been improved for solving TUSP with multiple shunting yard layouts and that planners are indeed capable of handling the complexities that come along with multiple shunting yard layouts.

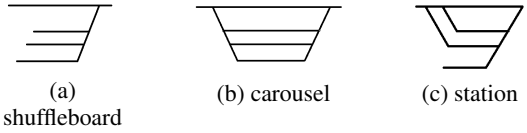
1 Introduction

The Dutch Railways is a very complex system with many trains that need to be operated during the day and stored at night. The planning and managing for this storing is done on shunting yards and is proven to be an NP-hard problem, also known as the Train Unit Shunting Problem or TUSP [1]. TUSP is a problem that focuses on matching arriving and departing trains and parking these on the correct shunting tracks, such that the costs for moving and parking are minimal. While the TUSP consists of many sub-problems such as servicing, matching and parking, the focus of this research is on the parking, routing and matching sub-problems.

To help human planners create a suitable plan for each night, a planning language called Planning Domain Definition Language or, PDDL, will be used for this research [2; 3]. PDDL consists of two main parts, the domain and the problem instance. The first describes the rules and components of a problem and the latter gives details on what the problem is.

While existing works that attempt to solve TUSP already exist, various assumptions about the shunting yards are made [1; 4]. Most existing works currently focus on the TUSP with one type of shunting yard rather than including all types in the problem. These analyses assume that the shunting yard is either a carousel, which consists of mostly free tracks, a shuffleboard, which consists of mostly Last In First Out, or LIFO, tracks, or a station, which is a hybrid layout without clear characteristics containing both free and LIFO tracks [5]. Additionally, no existing work tackles TUSP using PDDL, even though a specialized language for planning can provide great results when attempting to solve TUSP problems.

While LIFO tracks sound more restrictive compared to free tracks when considering the arrival and departure times of trains, as LIFO tracks can only be entered and left from one side whilst free tracks can be entered and left from both sides. Free tracks have a similar restriction when 3 or more trains are involved, limiting the trains in the middle [6]. Examples of these shunting yards are found below.



(a) shuffleboard (b) carousel (c) station

Figure 1: example tracks

To address the current research gap of lesser consideration of multiple shunting yard layouts in the TUSP and the solving of TUSP in PDDL, PDDL will be used to solve a basic version of the TUSP which includes carousels, stations and shuffleboards. Thus, this research aims to answer the research question: ***Can a planner in PDDL be optimized, which handles domains that contain commonly encountered types of shunting yard layouts?***

To answer the research question, the following sub-questions are defined:

1. What are the similarities and differences between different shunting yard layouts for PDDL?
2. Why should planners be able to generalize for domains with different shunting yard layouts?
3. How can current planners be used to solve domains with different shunting yard layouts?
4. How can the planner that is able to solve for different shunting yard layouts be optimised?

As including all types of shunting yard layouts would resemble real-life scenarios even more than current research, the main contributions of this paper are:

- development of a domain with appropriate predicates, actions and constraints for TUSP with multiple shunting yard layouts in PDDL,
- evaluation of planners' performance and effectiveness on the specified domain and problem instances,
- improvement of a chosen planner which can handle the specified domain,
- possible integration into real-world scenarios

The main insights of this research consist of the following: PDDL can be used to represent sub-problems of the TUSP and possibly TUSP as a whole. Suitable planners for solving sub-problems of the TUSP using PDDL might already exist, but need to be further analyzed and improved for better performances. Finally, real-world integration of using planners

and PDDL is possible, but the current domain needs to be further adapted to represent real-world scenarios since we would consider many more problems.

2 Background

In this section, background information will be provided for PDDL and planners, as well as a discussion of relevant literature on TUSP. In the following sub-sections, firstly relevant literature will be discussed. Second, basic information on PDDL will be given and finally, planners will be explained.

2.1 Relevant Literature

TUSP consists of multiple sub-problems which all need to be solved. These sub-problems are defined by Trepat [5] and are the following:

- **Matching**; arriving and departing train units need to be matched
- **Servicing**; tasks such as cleaning need to be performed at specific locations
- **Parking**; trains need to be parked, with an unobstructed route when departing
- **Splitting and combining**; splits and combinations are necessary to change train units
- **Routing**; trains need to be routed from stations to shunting yards
- **Crew scheduling**; crews are necessary to perform all tasks

Along with that, authors that have researched solutions for TUSP have already proven that TUSP is NP-hard [1; 7], which proves that TUSP is an extremely complex problem.

Authors have considered many approaches to solving the TUSP, one of which was exact solution approaches. Authors such as Lentink et al. [8], try to solve a decomposed version of TUSP in four steps. But exact solution approaches struggle with computation time, making them unsuitable to solve the TUSP.

Apart from exact solution approaches, heuristic approaches were explored by many. They were first introduced by Haeijema et al. [9], who were inspired by dynamic programming. Their solution created smaller sub-problems that were solved, to finalize solutions. Others have improved upon the solution from Haeijima et al., for example by adding a greedy heuristic along with the dynamic programming [10]. The most promising approach is currently provided by van den Broek et al. [11], with a local search approach that is successful in providing solutions for real-world scenarios.

However, almost none of the authors have tried to solve the TUSP using PDDL. Except for a few such as Cardellini [12], who attempted to solve a problem called the In-Station Train Dispatching Problem, which is similar to TUSP, using PDDL+ [13].

2.2 PDDL

The Planning Domain Definition Language [2], [3] is designed for task planning and can be used to solve and define a variety of planning problems. PDDL consists of two main components, namely the domain and the problem.

The domain defines the components and rules of a planning problem. This domain consists mainly of *types*, which define the class of an object. *Predicates*, which are properties of objects that are either true or false. *Actions* with a precondition and an effect, which are the allowed steps to solve the planning problem, and *functions*, which are used to define numeric quantities in the domain. Thus, the domain serves as a blueprint for formal planning problems. As an example, a domain with a robotic arm with an action to pick up a cupcake is provided.

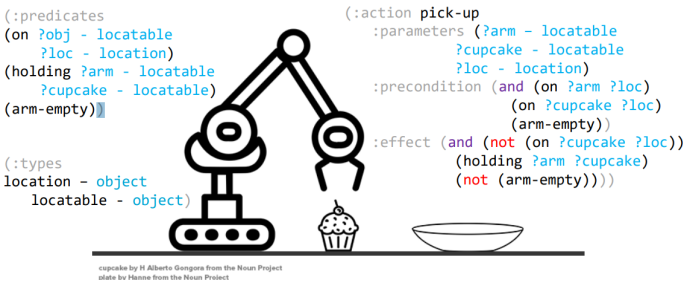


Figure 2: Example robotic arm domain

Problem instances are created from the domain by specifying objects, initial conditions and goals. These objects are specific instances of the types defined in the domain and represent the relevant entities in this problem. The initial conditions describe the initial state and goal defined by the predicates. For example, where a train is arriving from initially. And finally, the goal represents the desired state for a problem. in our case, the departing sequence. When the goal state is reached, the problem would be considered solved.

2.3 Planners

Planners can be seen as solvers and are commonly problem-independent. They tackle problem instances based on the domain description. This allows domains to change and evolve, whilst the planners are still usable. Different planners might use different algorithms to solve problems, which results in different speeds and costs for found solutions. Along with that, planners might have different goals for a solution. Some planners only attempt to find any solution, these are called satisficing planners. Others, search for an optimal solution with the lowest cost for a problem and are known as optimal planners [14].

3 Problem Description

In this section, the formal problem description of the Train Unit Shunting Problem for this thesis will be explained along with the definition of the domain that will be used. Although TUSP contains many problems that were defined in section 2, this paper only considers parts of the parking and routing problem with possible elements of the matching problem. In the next sub-sections, firstly the TUSP will be explained for this research. Afterwards, the problem will be defined in PDDL. And finally, the relevant TUSP sub-problems will

be explained and an example of a problem instance will be given.

3.1 TUSP Problem Definition

The Train Unit Shunting Problem for this research is defined as the following: Given a set of trains T with their arriving order A on train tracks v, is there a sequence of steps that allows all trains T to be in their departing order D after being parked $\forall t \in T; parked(t)$ on shunting yards S with shunting tracks $u \in S$, thus $parkedOn(t, u)$? These shunting yards are the shuffleboard S_T, the carousel S_C and the station S_S. This is the TUSP problem to solve for this research and even though the aim is not to mathematically solve this problem, it is helpful to understand the formal definition of this problem which helps explain certain concepts.

3.2 TUSP in PDDL

Now that TUSP has been formally defined, it needs to be translated to PDDL. Thus, the set $\{T, A, D, S, u, v\}$ needs to be defined in PDDL. As mentioned in Section 2, all components and rules in the domain will be based on this set. Therefore, the domain and problem instances need to be able to define the set $\{T, A, D, S, u, v\}$. All trains T, tracks u and v can be defined as types in the domain. The arrival A and departing order D can be given in the initial state and goal state defined by the problem. Shunting yards do not have to be defined explicitly in the domain, since no specific actions are needed for different shunting yards. Finally, having a train parked $parked(t)$ as well as where a train is parked $parkedOn(t, u)$ can be created in the predicates of the domain so that if a train is parked it will be set as true and where a train is parked is true for that specific train and track. The intricacies of this problem will be further analyzed in section 4, along with the created domain in section 5.

3.3 TUSP Sub-problems

TUSP contains many sub-problems as mentioned in section 2.1. However, not all sub-problems are relevant for this research, since we only focus on multiple shunting yard layouts. Thus, for example, the servicing sub-problem would not be helpful to solve for this research as it does not provide any help with multiple shunting yard layouts. The following sub-problems are relevant to this research and are the following: the parking problem, the routing problem and the matching problem.

Parking Problem

The parking problem consists of trains parking on train tracks so that all trains have been parked in the shunting yard, whilst allowing all trains to have an unobstructed path to their destination when leaving. The parking problem is the main problem that this research attempts to solve since this problem is most relevant when using different shunting yard layouts. For example, free tracks need to take into account the side of arrival, as well as the time of arrival. While LIFO tracks only consider the time of arrival. To allow more complex plans to be solved, it is allowed to switch train units between tracks as long as they will be parked on a track in the final plan. This makes it possible to shuffle train units onto other tracks to free up space for other trains to arrive or leave.

Routing Problem

The routing problem is similar to the parking problem, where the goal is to find an unobstructed path to their destination while attempting to minimize the time taken. Although finding the optimal solution with the minimum time taken for problems is unnecessary for this research, it is a metric on which the planners will be evaluated on. Planners that are closer to the optimal solution will be evaluated higher than planners that create solutions with higher costs.

Matching Problem

For the matching problem, a set of arriving trains is given as well as a set of departing trains, with arriving and departing times respectively. The matching problem also matches different train units from arriving trains to certain train units from departing trains. This latter is, however, not relevant for the specified domain, since the domain only contains trains with one train unit and thus each arriving train would result in the same departing train.

3.4 Example Problem Instance

In this section, an example of a problem instance with both LIFO and free tracks is given, to clarify the TUSP in PDDL that this research attempts to solve. This problem instance is also used for the evaluation of the planners.

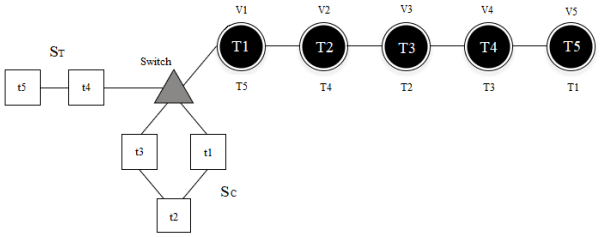


Figure 3: Example Problem of TUSP

As seen in Figure 3, Each train track v_i is a black node along with the trains T_i inside to depict the starting order. Additionally, the trains T_k under the nodes would provide the departing sequence of the trains, which dictates the position all trains should be in for the goal state. Further left from that, a switch can be observed, a special track $t0$ that connects the starting train tracks and shunting tracks, as well as connecting more than two tracks, connecting shunting yards S_C and S_T by their shunting tracks t_i.

An optimal solution in terms of the cost would cost 46 steps. Note that more than one optimal solution is possible since other solutions can have the same cost. An example optimal solution with the total steps on the right is provided in Figure 4.

1. T1 moves to t1 (2)
2. T2 moves to t2 (6)
3. T3 moves to t3 (10)
4. T4 moves to t5 (16)
5. T5 moves to t4 (22)
6. T1 moves to v5 (28)
7. T3 moves to v4 (33)
8. T2 moves to v3 (38)
9. T5 moves to t1 (40)
10. T4 moves to v2 (44)
11. T5 moves to v1 (46)

Figure 4: Example optimal solution

4 Methodology

This section provides the steps used for this research along with our contributions, which will be further described in detail in the Results section hereafter. We have first created a suitable domain to represent the TUSP with problem instances for this research. After that, we evaluated the planners on certain problem instances. Finally, we made optimisations of the planner by altering the domain. An overview of the steps taken and contributions made for this research will be given in these three separate sub-sections.

4.1 Creating the Domain and the Problems

Having a correct domain is necessary to create effective problem instances and to evaluate planners properly. The domain has drastic consequences for this research, as it can influence the solvability of a problem and the speed at which planners can solve a problem. As defined in section 3.2, the domain should contain trains, train tracks and support for different shunting tracks, as well as predicates such as *parked* for trains and actions that allow trains to move and park on tracks.

The domain

Our contributions to the finalized domain were the addition of LIFO and free tracks in the types and the action to switch between shunting yards. As for this research, a domain was given by the responsible professor, on which the final domain and problem instances are built. The given domain already satisfied all the necessary attributes and only needed to be extended to solve specific issues regarding different shunting yards.

Although the finalized domain is quite basic, there were other extendable possibilities considered that were complex. One such, was the idea of creating LIFO tracks and free tracks within the domain, by creating single-ended queues and double-ended queues in the domain with the use of *derived predicates*.

This option, however, was not chosen since we believed it to be too time-consuming and out of the scope of this research. Having such tracks in the domain had the possibility of simplifying problem instances since individual tracks would not be needed to be defined, but this was not the goal of this research. We have, however, chosen to keep LIFO and free track types in the domain to clarify what tracks are within the problem instances. But actions for different track types were not in the domain and LIFO and free tracks were represented in the problem instances.

Finally, switches were used in the domain, which allows trains to go from one train track to multiple others. Due to the limitations of the original actions used for switches, we have created an additional action to allow trains to shuffle between LIFO and free tracks. And even though allowing trains to move between switches that were connected or tracks in between switches was a possibility that was considered, they were deemed redundant since both situations could be avoided by merging both incoming and outgoing tracks from switches into one switch in the problem instances.

Creating the Problems

In addition to the domain, problem instances were needed to compare, evaluate and optimise planners. Our contributions to the problem instances were nine different problems, in three different categories: simple, medium and complex. These were also created in the three different shunting yard layouts that we have found, namely the Shuffleboard layout, the Carousel layout and the Station layout. They were chosen to determine how well planners can solve the different problem instances and differentiate between shunting yard layouts. These problem instances were categorized by how many trains were present in the problems and which tracks were used. For these problems three trains were considered simple, five trains were medium and ten trains were a complex problem. These numbers are fairly arbitrary since the number of trains is not the only determining factor of complexity for these problems. But as these problems are all created by hand with only a few possible optimal solutions, we deemed them suitable for this research.

4.2 Evaluating Planners

The development of a planner which supports multiple shunting yard layouts is the goal of this research. To find a suitable planner for this research, planners need to be evaluated based on certain metrics. For this research, planners in the satisficing track from the International Planning Competition from 2018 were chosen [14]. These planners were chosen due to their availability on the mapfw server, to speed up the research process by avoiding the time to set up new planners, even though other planners online could have been more suitable for this problem. Finally, the chosen planners that were going to be evaluated were the *Baseline-explicit-planner*, *Team2*, *Team4* and *Team35*. These planners can all be found in the International Planning Competition from 2018.

The planners would finally only be evaluated on complex problem instances since simple and medium problems did not provide enough information on the performance of many of the planners. One such planner would always take a minimum of two minutes to finish, no matter how simple the problem was.

The planners would be evaluated and compared based on three metrics. These were solvability, speed and plan cost. The solvability metric would provide information on how many problem instances the planner was able to solve. Speed would be measured by how long it takes for the planner to finish running in seconds. And plan cost is the cost of the plan found. Even though solvability is the main concern for this research, other metrics could provide better insights into how a planner is performing with multiple shunting yard layouts.

Do note that we have a timeout of 20 minutes for speed. We wanted planners to solve problems within 10 minutes but allowed some opportunities for planners that could not finish within this time frame. Additionally, this speed is only an estimate since it is determined by the speed on the server which could be influenced by the amount of users. The speed measured is the average over three attempts, to minimise the error.

4.3 Optimising Planners

Based on the metrics that are found when evaluating the planners, one planner would be chosen to be further optimised. This optimisation could be made in many different ways, some possibilities are the modification of code within the planner or changing the domain to get better scores in one or more of the three metrics.

Our contribution to the optimisation of the planner was mainly done by optimising the domain. Due to the chosen planner being extremely hard to modify with no guarantee of optimisations being possible, the domain was updated in an attempt to create better performances for the chosen planner, either by speeding up the plan or reducing cost. The contributions to the updated domain include differentiating LIFO tracks and free tracks, by having new predicates and different actions for both. Free tracks would behave the same, but LIFO tracks would have more information such as the last shunting track where a train can park. The change to only update LIFO tracks was made, due to the results found from the evaluation of planners, where the chosen planner behaved much worse in certain problems with LIFO tracks.

Finally, we have created an extra problem with only LIFO tracks to confirm the improvements that were made with LIFO tracks. This problem is created similarly to the original Shuffleboard problem instance and is evaluated the same as the other problems.

5 Results

In this section, the results of the finalized domain, the evaluation of planners and the optimisation for the planner will be given. Firstly, the finalized domain will be provided. Secondly, the results of the evaluation of the planners will be presented along with which planner was chosen to be optimised. Finally, the result of the optimisation can be found, with the optimisation itself.

5.1 The Finalized Domain

The result of the finalized domain is not that different from the original domain that was provided. The key differences are mainly found in the problem instances, where both LIFO and free tracks would be created. The full domain can be found in Appendix A.

In the domain, first, the types were defined. In the domain types, *Trackpart* refers to individual trackparts, *track* are the tracks in the domain and *LIFO* and *free* are the types of tracks. Additionally, *track* is meant to mean a shunting track and *trainunits* are individual trains.

After that, the predicates are defined. These predicates were not modified from the provided domain and should be self-explanatory. The most important predicate that was used was the *nextTo* predicate. This was the predicate that was the core of creating LIFO tracks and free tracks in PDDL. LIFO tracks can be defined in the problem, by connecting tracks to create a line, with the last track having only one track connected. By connecting tracks using *nextTo*, free tracks can also be defined. With no last track and circling back to the starting track, all tracks have two tracks next to them. An example of a free track can be found below, where the PDDL plugin for Visual Studio Code provides a visualisation of the problem instance in Figure 5.

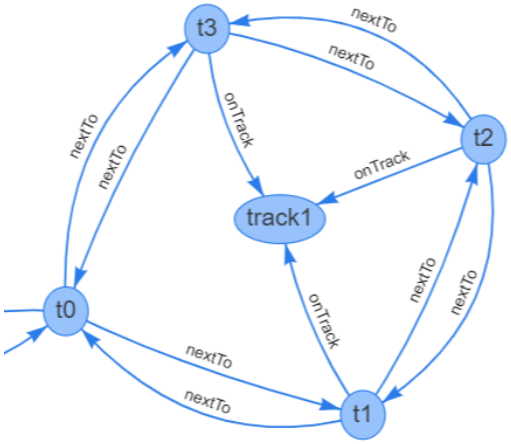


Figure 5: Visualisation of a free track using Visual Studio Code

Finally, actions were defined. These actions allowed trains to move from one track to another track and made sure that trains were parked. As an example in Figure 6, actions such as *move-along-track* moved trains between shunting tracks after being parked.

```
; action to move a trainunit along a track
(:action move-along-track
    :parameters (?train - trainunit ?from ?to - trackpart ?t - track)
    :precondition (and (at ?train ?from) (free ?to)
                       (nextTo ?from ?to) (onTrack ?from ?t)
                       (onTrack ?to ?t))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                       (free ?from) (not (free ?to)))
)
```

Figure 6: Move-along-track action used in the domain

Do note, however, that the actions differentiate between different types of train tracks. These are namely the shunting tracks, where trains can be parked, the arrival or departure tracks, where trains start from and depart and the switch tracks, which connect more than two train tracks next to them. Thus different actions were necessary to distinguish what track is used.

Additionally, there is one action called *switch-track* in Figure 7, which allowed trains to pass over a switch to park on a different shunting track. This was created to differentiate between trains leaving the shunting yard and trains simply going to the switch to change tracks.

```
; action to move a trainunit to out of a track
; and reset the parkedOn predicate
; used for shuffling trains to different tracks
(:action switch-track
    :parameters (?train - trainunit ?from ?switch ?to - trackpart
        ?t1 ?t2 - track)
    :precondition (and (at ?train ?from) (free ?to) (free ?switch)
                    (nextTo ?from ?switch) (onTrack ?from ?t1)
                    (switch ?switch) (nextTo ?switch ?to)
                    (onTrack ?to ?t2) (not (forall (?unit - trainunit)
                    (hasBeenParked ?unit))))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                    (free ?from) (not (free ?to))
                    (not (parkedOn ?train ?t1))
                    (parkedOn ?train ?t2))
)
```

Figure 7: Switch-track action used in the domain

Finally, the problem instances that were created in three categories, all had the same type of problem that needed to be solved. For each category, one problem would contain a shuffleboard layout, with only LIFO tracks. Another problem would contain a carousel layout, with only free tracks. And finally, the last problem would have the station layout, having a mix of LIFO and free tracks. For illustration purposes, the complex problem instances are shown below in Figures 8-10.

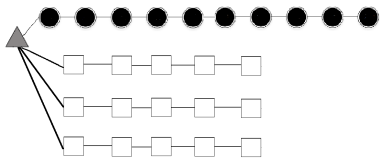


Figure 8: Complex Shuffleboard problem

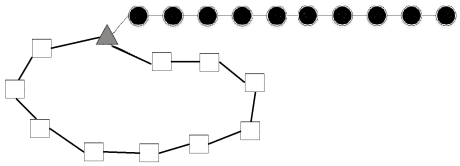


Figure 9: Complex Carousel problem

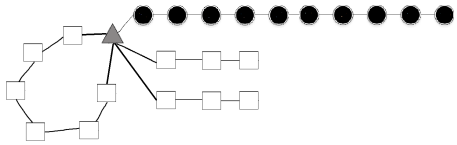


Figure 10: Complex Station problem

5.2 Evaluation of the Planners

The evaluation of these planners was based on three metrics, as mentioned in section 4. It was decided to use only the complex problems that were found in Figure 8-10 as a final metric, as this would replicate real-world scenarios the most. The final results of the complex problems are found in the following Table 1 with speed in time, cost in steps and solvability as a boolean.

	Shuffleboard	Carousel	Station
Baseline	1200s N/A False	1200s N/A False	1200s N/A False
Team2	1200s N/A False	1200s N/A False	1200s 73 True
Team4	182s 213 True	2.6s 199 True	3.2s 185 True
Team35	336s N/A False	323s N/A False	296s 173 True

Table 1: Results of complex problems based on speed, cost and solvability

Some interesting findings from the results in Table 1 are that the speed for the *Baseline* and *Team2* is always 1200 seconds. This is due to the 20-minute timeout that was chosen. Apart from that, even though *Team2* times out for the Station problem, it can find a plan within the time limit but does not finish running. Finally, the only planner that was able to solve all problems was *Team4*, with extremely high speeds for both the Carousel problem and the Station problem.

From this, it can be determined that *Team4* was the most suitable planner for different shunting yard layouts and it was chosen to be further improved for optimisation.

5.3 Optimisation for the Team4 Planner

The strategy that *Team4* uses to solve problem instances, is to translate them into boolean satisfiability problems [15]. *Team4* is known as an SAT solver and would be very hard to optimise itself by changing either the translation process from PDDL to SAT or the SAT encoding of *Team4*. Thus the domain was updated for optimisation and to create better performances.

As seen in Table 1, *Team4* struggled with finding a plan in a short amount of time for the Shuffleboard problem. Thus, the focus for this optimisation would be on LIFO tracks in the domain.

Although a difference between LIFO and free tracks was already made in the types, it would still need to be differentiated in the actions. Thus a simple predicate *free-track* was created to determine if a track was a free track. As an optimisation, trains on LIFO tracks would only move to the last possible track and needed separate actions for this. To make sure that the last track was correct, the original *nextTo* predicate was changed to two predicates, namely *next* and *prev* found in Figure 11.

Naturally, the problem instances were also changed to use the newly optimised predicates and actions. The fully optimised domain can be found in Appendix B.

```
(:predicates
    (next ?x ?y - trackpart) ;track part x next to track part y
    (prev ?x ?y - trackpart)  ;track part x previous from track part y
    (onTrack ?x - trackPart ?y - track) ;track part x on track y
    (at ?x - trainunit ?y - trackpart) ;train unit x on track part y
    (hasBeenParked ?x - trainunit) ;true if x is parked on some track
    (free ?x - trackpart) ;trackpart x has nothing parked there
    (parkedOn ?x - trainunit ?y - track) ; indicates x parked on track y
    (onPath ?x) ;trackpart x is on the arrival/departure path L
    (switch ?x) ;trackpart x is a switch
    (free-track ?x - track)
    (last-track ?x - trackpart)
)
```

Figure 11: Predicates of the optimised domain

The improved actions that were used, moved trains from the switch to the *last-track* or moved the train that was parked before the *last-track* back to the switch. Other actions included the switching of LIFO tracks and separate actions for when the *last-track* was next to the switch. Additionally, the original actions for the shunting yard in the domain yard would only be applicable for free tracks due to the usage of the new predicate *free-track*. Free tracks are thus unable to use the LIFO actions, since the *last-track* predicate would not be defined for free tracks.

To analyze the effectiveness of the LIFO track changes even further, a new problem named *Shuffle2* was created. This problem was created similarly to the complex Shuffleboard problem, to confirm the improvement of LIFO tracks in the domain. This new problem can be seen in Figure 12.

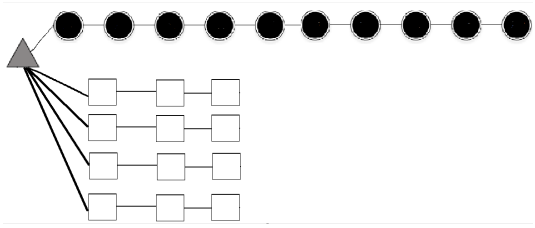


Figure 12: Additional created Shuffleboard problem

	Shuffleboard	Carousel	Station	Shuffle2
Team4	182s 213 True	2.6s 199 True	3.2s 185 True	47.5s 170 True
	Improved	Improved	Improved	Improved
Team4	6.4s 145 True	3s 193 True	10s 176 True	5.4s 151 True

Table 2: Results of Team4 before and after improvements, of complex problems based on speed, cost and solvability

As a result of the optimisation by differentiating LIFO tracks and free tracks in the domain, *Team4* would have significant speed-ups in the Shuffleboard problems and some minor plan costs would be improved.

For all problems, the cost of the plan had been reduced and all problems remained solvable. However, the speed for both the Carousel and Station problem decreased slightly. Interestingly, the cost for the Carousel problem decreased even though it contained free tracks exclusively, whilst only LIFO tracks were changed. we believe this would be due to how Team4 translates the SAT problem and how different propositions are now created which could influence the translation time.

All in all, as can be seen in Table 2, a decent improvement was made for *Team4* by improving the speed for shuffleboard layouts significantly and reducing plan costs for all problem instances.

6 Responsible Research

The research conducted in this thesis is centered around creating planners in PDDL to utilize different shunting yard layouts. While the primary objective is to enhance the efficiency and effectiveness of planners, we have to acknowledge and address the ethical aspects and responsible considerations with this research.

During this research, no contact with humans and sensitive data was made. Thus there are no relevant ethical aspects for this research. For reproducibility, the domain can be found in Appendix A and the improved domain can be found in Appendix B. Problem instances have illustrations throughout the thesis, namely Figures 3, 8, 9, 10 and 12. Even though not all readers would have access to the mapfw server of TU Delft, the planners are still available online [14] and can be used to yield the same or similar results as the ones found in section 5.

7 Conclusions and Future Work

In this research, we addressed the use of planners in PDDL to handle different shunting yard layouts in the TUSP. The objective was to develop a planner that can create suitable plans for created problem instances with different shunting yards.

To fill the research gap of solving TUSP using primarily one shunting yard type, we developed a domain in PDDL that contains the commonly encountered types of shunting yard layouts, including carousels, shuffleboards, and stations. We evaluated planners capable of handling this domain, by assessing its performance and effectiveness on different problem instances. The chosen *Team4* planner was further optimised, by changing the domain accordingly to promote better performances for shuffleboards.

Our contributions in this paper include the development of appropriate predicates, actions and constraints for the TUSP with multiple shunting yard layouts in PDDL. Along with the improvement of a planner for this domain from the IPC2018, and the evaluation of its performance.

The results obtained from our experiments provide insights into the planner's effectiveness and its potential integration into real-world scenarios.

Additionally, this research expands the scope of shunting yard layouts used in the TUSP. By addressing the limitations of existing works and utilizing PDDL, we have demonstrated that planners can handle the complexities associated with different shunting yard layouts.

In conclusion, we have successfully answered the research question posed in this study: ***Can a planner in PDDL be improved which handles domains that contain commonly encountered types of shunting yard layouts?***. The findings from this research provide a foundation for further advancements in the optimization of the TUSP for PDDL, ultimately integrating the use of PDDL with real-world scenarios.

7.1 Future Work

While this research has made progress in addressing the optimization of planners for different shunting yard layouts, many improvements can still be made.

Improvements such as implementing queues to represent LIFO and free tracks can still be implemented within the domain, which could result in even more speedups and lower plan costs.

Furthermore, evaluations can be expanded by incorporating larger problem instances with 20-30 trains that represent real-world scenarios. On top of that, the chosen *Team4* planner might still allow for more improvements if further analyzed.

And finally, alternative planners that were not part of the International Planning Competition could be explored, which may result in an even better planner optimised for different shunting yard layouts.

A The Modified Domain used for this research

```
(define (domain domain1)
(:requirements :adl)
(:types
    trackpart track trainunit - object
    ; these are the different types of train units
    icm virm sng slt - trainunit
    LIFO free - track
)
(:predicates
    (nextTo ?x ?y - trackpart) ;track part x next to other track part y
    (onTrack ?x - trackPart ?y - track) ;track part x on track y
    (at ?x - trainunit ?y - trackpart) ;train unit x on track part y
    (hasBeenParked ?x - trainunit) ;true if x is parked on some track
    (free ?x - trackpart) ;trackpart x has nothing parked there
    (parkedOn ?x - trainunit ?y - track) ; indicates x parked on track y
    (onPath ?x) ;trackpart x is on the arrival/departure path L
    (switch ?x) ;trackpart x is a switch
)
; action to move a trainunit to a neighbouring trackpart
; on a track, to park it
(:action move-to-track
    :parameters (?train - trainunit ?from ?to - trackpart ?t - track)
    :precondition (and (at ?train ?from) (free ?to)
                    (nextTo ?from ?to) (onTrack ?to ?t)
                    (switch ?from))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                    (free ?from) (not (free ?to))
                    (hasBeenParked ?train) (parkedOn ?train ?t))
)
; action to move a trainunit to out of a track
; and reset the parkedOn predicate
; used for shuffling trains to different tracks
(:action switch-track
    :parameters (?train - trainunit ?from ?switch ?to - trackpart
        ?t1 ?t2 - track )
    :precondition (and (at ?train ?from) (free ?to) (free ?switch)
                    (nextTo ?from ?switch) (onTrack ?from ?t1)
                    (switch ?switch)
                    (nextTo ?switch ?to) (onTrack ?to ?t2)
                    (not (forall (?unit - trainunit)
                    (hasBeenParked ?unit))))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                    (free ?from) (not (free ?to))
                    (not (parkedOn ?train ?t1))
                    (parkedOn ?train ?t2))
)

; action to move a trainunit along a track
(:action move-along-track
    :parameters (?train - trainunit ?from ?to - trackpart ?t - track)
    :precondition (and (at ?train ?from) (free ?to)
                    (nextTo ?from ?to) (onTrack ?from ?t)
                    (onTrack ?to ?t))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                    (free ?from) (not (free ?to)))
)
; Can only move back to departure if all trains have been parked.
(:action move-to-departure
    :parameters (?train - trainunit ?from ?to - trackpart)
    :precondition (and (at ?train ?from) (free ?to)
                    (nextTo ?from ?to) (onPath ?to)
                    (forall (?unit - trainunit) (hasBeenParked ?unit)))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                    (free ?from) (not (free ?to)))
)
; Action to move train unit over the arrival path
; towards the shunting yard
(:action move-on-arrival
    :parameters (?train - trainunit ?from ?to - trackpart)
    :precondition (and (at ?train ?from) (free ?to)
                    (nextTo ?from ?to) (not (hasBeenParked ?train))
                    (onPath ?from))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                    (free ?from) (not (free ?to)))
)
; Distinguish from switch-track
(:action move-from-track-to-departure
    :parameters (?train - trainunit ?from ?to - trackpart ?t - track)
    :precondition (and (at ?train ?from) (free ?to)
                    (nextTo ?from ?to) (onTrack ?from ?t)
                    (switch ?to)
                    (forall (?unit - trainunit) (hasBeenParked ?unit)))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                    (free ?from) (not (free ?to))
                    (not (parkedOn ?train ?t)))
)
)
```

B The Improved Domain

```
(define (domain imprDomain)

(:requirements :adl)

(:types
    trackpart track trainunit - object
    icm virm sng slt - trainunit
        ; these are the different types of train units
    LIFO free - track
)

(:predicates
    (next ?x ?y - trackpart) ;track part x next to track part y
    (prev ?x ?y - trackpart)  ;track part x previous from track part y
    (onTrack ?x - trackPart ?y - track) ;track part x on track y
    (at ?x - trainunit ?y - trackpart) ;train unit x on track part y
    (hasBeenParked ?x - trainunit) ;true if x is parked on some track
    (free ?x - trackpart) ;trackpart x has nothing parked there
    (parkedOn ?x - trainunit ?y - track) ; indicates x parked on track y
    (onPath ?x) ;trackpart x is on the arrival/departure path L
    (switch ?x) ;trackpart x is a switch
    (free-track ?x - track)
    (last-track ?x - trackpart)
)

; action to move a trainunit to a neighbouring
; trackpart on a track to park it
(:action move-to-track
    :parameters (?train - trainunit ?from ?to - trackpart ?t - track)
    :precondition (and (at ?train ?from) (free ?to)
                    (or (next ?from ?to) (prev ?from ?to))
                    (onTrack ?to ?t) (onTrack ?to ?t)
                    (free-track ?t) (switch ?from))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                    (free ?from) (not (free ?to))
                    (hasBeenParked ?train) (parkedOn ?train ?t))
)

; action to move a trainunit to out of a track,
; and reset the parkedOn predicate
; used for shuffling trains to different tracks
(:action switch-track
    :parameters (?train - trainunit ?from ?switch ?to - trackpart
        ?t1 ?t2 - track )
    :precondition (and (at ?train ?from) (free ?to) (free ?switch)
                    (or (next ?from ?switch) (prev ?from ?switch))
                    (onTrack ?from ?t1) (switch ?switch)
                    (free-track ?t1) (free-track ?t2)
                    (or (next ?switch ?to) (prev ?switch ?to))
                    (onTrack ?to ?t2) (not (forall (?unit - trainunit)
                    (hasBeenParked ?unit))))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                    (free ?from) (not (free ?to))
                    (not (parkedOn ?train ?t1))
                    (parkedOn ?train ?t2))
)

; ; action to move a trainunit along a track
(:action move-along-track
    :parameters (?train - trainunit ?from ?to - trackpart ?t - track)
    :precondition (and (at ?train ?from) (free ?to)
                    (or (next ?from ?to) (prev ?from ?to))
                    (onTrack ?from ?t) (onTrack ?to ?t)
                    (free-track ?t))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                    (free ?from) (not (free ?to)))
)
; Can only move to track if all trains have been parked
; to distinguish from switch-track
(:action move-from-track-to-departure
    :parameters (?train - trainunit ?from ?to - trackpart ?t - track)
    :precondition (and (at ?train ?from) (free ?to)
                    (or (next ?from ?to) (prev ?from ?to))
                    (onTrack ?from ?t) (switch ?to) (free-track ?t)
                    (forall (?unit - trainunit) (hasBeenParked ?unit)))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                    (free ?from) (not (free ?to))
                    (not (parkedOn ?train ?t)))
)
; switches between LIFO tracks
(:action switch-LIFO-track
    :parameters (?train - trainunit
        ?from ?prev ?switch ?newLast ?last - trackpart ?t1 ?t2 - track)
    :precondition (and (at ?train ?from) (onTrack ?from ?t1)
                    (onTrack ?last ?t2) (prev ?from ?prev)
                    (prev ?last ?newLast) (free ?switch)
                    (switch ?switch) (last-track ?prev)
                    (last-track ?last)
    )
    :effect (and (at ?train ?last) (not (at ?train ?from))
                    (free ?from) (last-track ?from) (not (last-track ?prev))
                    (not (last-track ?last)) (last-track ?newLast)
                    (not (parkedOn ?train ?t1)) (parkedOn ?train ?t2)
                    (not (free ?last))
    )
)
; action to move to switch from a LIFO track
(:action move-to-switch-LIFO-track
    :parameters (?train - trainunit ?from ?prev ?con ?switch - trackpart
        ?t - track)
    :precondition (and (at ?train ?from) (onTrack ?from ?t)
                    (onTrack ?con ?t) (prev ?from ?prev)
                    (onTrack ?prev ?t) (last-track ?prev)
                    (prev ?con ?switch) (free ?switch) (switch ?switch))
    :effect (and (at ?train ?switch) (not (at ?train ?from))
                    (free ?from) (not (free ?switch))
                    (not (last-track ?prev)) (last-track ?from)
                    (not (parkedOn ?train ?t)))
)
; action to move a train to the switch
; when last-track is next to the switch
(:action move-to-switch-LIFO-track-last
    :parameters (?train - trainunit ?from
?switch - trackpart ?t - track)
    :precondition (and (at ?train ?from) (onTrack ?from ?t)
                    (prev ?from ?switch) (switch ?switch)
                    (free ?switch) (not (free-track ?t)))
    :effect (and (at ?train ?switch) (not (at ?train ?from))
                    (free ?from) (not (free ?switch))
                    (last-track ?from)
                    (not (parkedOn ?train ?t)))
)

; action to move a trainunit to the last-track of a LIFO track
(:action move-to-LIFO-track
    :parameters (?train - trainunit ?from ?to ?prev ?last - trackpart
        ?t - track)
    :precondition (and (at ?train ?from) (free ?to)
                    (not (last-track ?to)) (or (next ?from ?to)
                    (prev ?from ?to)) (onTrack ?to ?t)
                    (switch ?from) (last-track ?last)
                    (prev ?last ?prev) (onTrack ?last ?t))
    :effect (and (at ?train ?last) (not (at ?train ?from))
                    (free ?from) (not (free ?last))
                    (not (last-track ?last)) (last-track ?prev)
                    (hasBeenParked ?train) (parkedOn ?train ?t))
)

; action to move a trainunit to the last-track
; that is next to the switch
(:action move-to-LIFO-track-onetrack
    :parameters (?train - trainunit ?from ?last - trackpart ?t - track)
    :precondition (and (at ?train ?from)
                    (or (next ?from ?last) (prev ?from ?last))
                    (onTrack ?last ?t) (switch ?from)
                    (last-track ?last))
    :effect (and (at ?train ?last) (not (at ?train ?from))
                    (free ?from) (not (free ?last))
                    (not (last-track ?last))
                    (hasBeenParked ?train) (parkedOn ?train ?t))
)
; Can only move back to departure if all trains have been parked.
(:action move-to-departure
    :parameters (?train - trainunit ?from ?to - trackpart)
    :precondition (and (at ?train ?from) (free ?to)
                    (or (next ?from ?to) (prev ?from ?to)) (onPath ?to)
                    (forall (?unit - trainunit) (hasBeenParked ?unit)))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                    (free ?from) (not (free ?to)))
)

; Action to move train unit over the arrival path
; towards the shunting yard
(:action move-on-arrival
    :parameters (?train - trainunit ?from ?to - trackpart)
    :precondition (and (at ?train ?from) (free ?to)
                    (or (next ?from ?to) (prev ?from ?to))
                    (not (hasBeenParked ?train)) (onPath ?from))
    :effect (and (at ?train ?to) (not (at ?train ?from))
                    (free ?from) (not (free ?to)))
)


)
```

References

[1] R. Freling, R. M. Lentink, L. G. Kroon, and D. Huisman, "Shunting of passenger train units in a railway station," *Transportation Science 39.2*, p. 261–272, 5 2005. [Online]. Available: https://doi.org/10.1287/trsc.1030.0076

[2] A. Green, B. J. Reji, ChrisE2018, and C. Muise, "Planning.wiki - the ai planning & pddl wiki," accessed on 2023-04-25. [Online]. Available: https://planning.wiki

[3] D. V. McDermott, "Pddl - planning domain definition language," accessed on 2023-5-26. [Online]. Available: http://www.cs.yale.edu/homes/dvm/

[4] I. Hanou, M. M. de Weerdt, and J. Mulderij, "Moving trains like pebbles: A feasibility study on tree yards," *Proceedings of the International Conference on Automated Planning and Scheduling*, to be published 2023.

[5] J. Trepat Borecka, "Routing optimization for the train unit shunting problem in a multi-agent deep reinforcement learning framework," 2021.

[6] N. C. L. van Bavel, "The application of a hybrid evolutionary algorithm to the train unit shunting problem."

[7] R. M. Lentink, "Algorithmic decision support for shunt planning," *Erasmus Research Institute of Management*, 2006.

[8] R. M. Lentink, P.-J. Fioole, L. G. Kroon, and C. van't Woudt, "Applying operations research techniques to planning of train shunting," *Planning in Intelligent Systems: Aspects, Motivations, and Methods*, p. 415–436, 2006. [Online]. Available: https://doi.org/10.1002/0471781266.ch15

[9] R. Haijema, C. W. Duin, and N. van Dijk, "Train shunting: A practical heuristic inspired by dynamic programming," *Planning in Intelligent Systems: Aspects, Motivations, and Methods*, pp. 437–475, 2006. [Online]. Available: https://doi.org/10.1002/0471781266.ch165

[10] M. van den Akker, H. Baarsma, J. Hurink, M. Modelski, J. Jan Paulus, I. Reijnen, D. Roozemond, and J. Schreuder, "Shunting passenger trains: getting ready for departure," *Proceedings of European Study Group Mathematics with Industry*, 2008.

[11] R. van den Broek, H. Hoogeveen, M. van den Akker, and B. Huisman, "A local search algorithm for train unit shunting with service scheduling," pp. 1–42, 2021.

[12] M. Cardellini, "Artificial intelligence techniques for solving the in-station train dispatching problem," *University of Genoa*, 2021.

[13] M. Fox and D. Long, "Modelling mixed discrete-continuous domains for planning," *University of Strathclyde*, 2006.

[14] F. Pommerening, A. Torralba, and T. Balyo, "Ipc2018," accessed on 2023-5-21. [Online]. Available: https://ipc2018-classical.bitbucket.io

[15] M. L.Littman, J. Goldsmith, and M. Mundhenk, "The computational complexity of probabilistic planning," *Journal of Artificial Intelligence Research 9*, pp. 1–36, 1998. [Online]. Available: https://doi.org/10.1613/jair.505