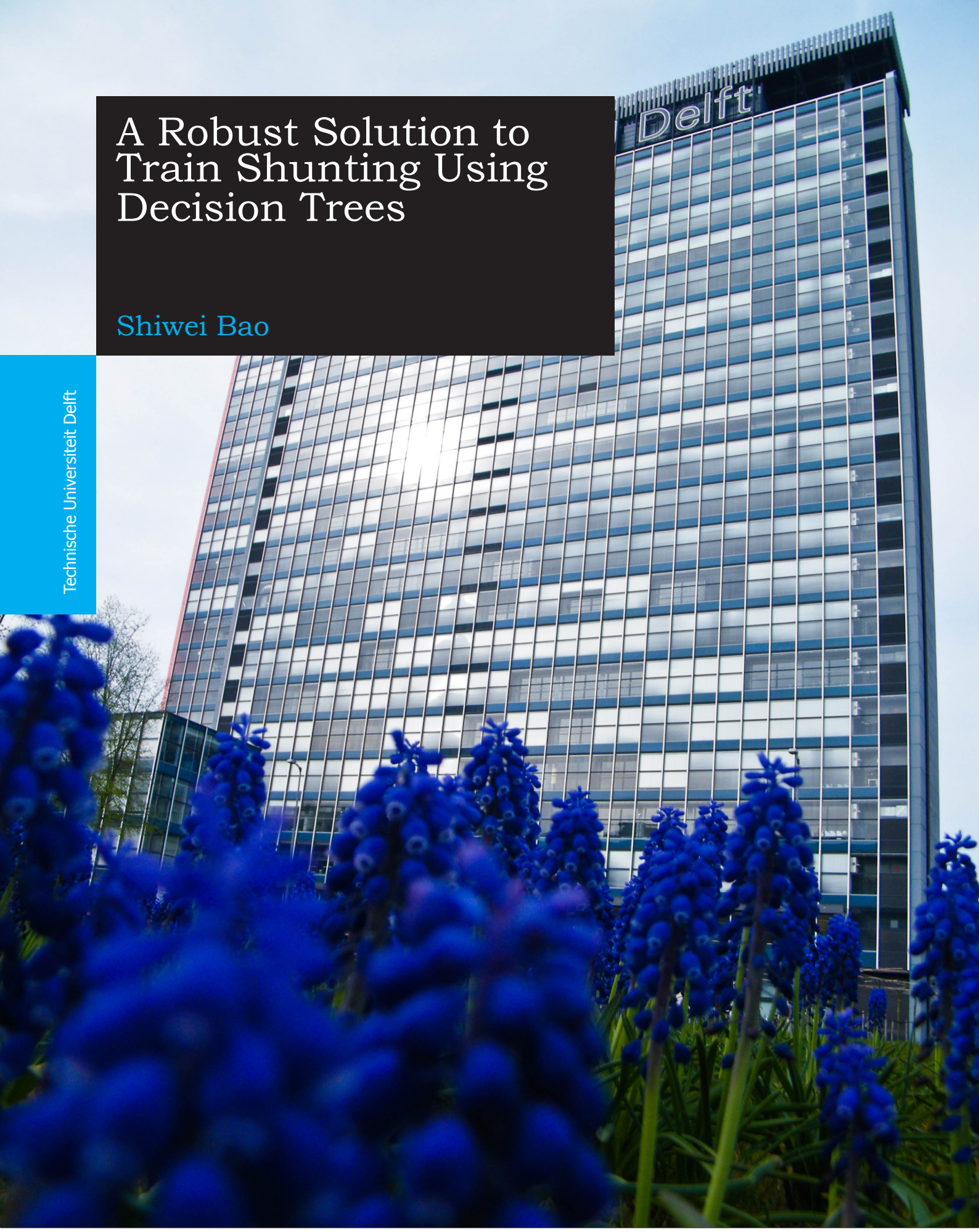
A Robust Solution to Train Shunting Using Decision Trees

Shiwei Bao

Technische Universiteit Delft



TU Delft Delft University of Technology

Challenge the future

A Robust Solution to Train Shunting Using Decision Trees

by

Shiwei Bao

in partial fulfillment of the requirements for the degree of

Master of Science
in Computer Science

at the Delft University of Technology,
to be defended publicly on 29th October, 2018 at 14:00 PM.

Supervisor:	Dr. ir. Sicco Verwer
	Dr. ir. Mathijs de Weerdt
	Dr. Wan-Jui Lee
Other thesis committee:	Dr. Jan van Gemert
	Dr. Laurens Bliek

This thesis is confidential and cannot be made public until October, 2018.

TU Delft Delft University of Technology

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science at the Delft University of Technology. I was engaged in researching and writing this thesis from January to October in 2018.

This project was performed at the request of NS Techniek, where I undertook an internship for the entire duration of my research. I was amazed by the efficient railway transportation on the first day I arrived in the Netherlands. Although, sometimes there are little delays. During the courses study, I found my interest in Machine Learning and wanted to apply this technique to support decision makings. That explains why I decided to undertake this project as my thesis which applies Machine Learning to the train shunting problem.

This thesis would not have been the same without the cooperation of many others. First and foremost, I would like to express my gratitude towards my supervisors, Mathijs de Weerdt and Sicco Verwer, for their excellent guidance and support in the past ten months. Their unwavering enthusiasm has supported me throughout my research, and their constructive criticism encouraged me to polish many parts of my thesis. Also, I am grateful to my manager Bob Huisman for offering this internship in Maintenance Development department. Many thanks to my mentor Wan-Jui Lee and my colleague Demian de Ruijter at NS Techiek for all the insightful discussions we have had on service site scheduling. Especially at the beginning of my thesis, this helped me remarkably to perceive the complexity of the problem. Furthermore, I appreciate the inspiring discussions we had in NS deep learning group and weekly master meetings in Delft. Last but not least, I would like to thank my parents for their mental and financial support during the project.

I hope you all enjoy reading this thesis.

Shiwei Bao
Delft, October 2018

Contents

1

Scientific Paper

A Robust Solution to Train Shunting Using Decision Trees

Shiwei Bao
Delft University of Technology
s.bao@student.tudelft.nl

Abstract

This research tackles the Train Unit Shunting Problem (TUSP) in train maintenance service sites. Many researches focus on producing feasible solutions, but only a few of them concentrate on the robustness of solutions. In reality, it is preferred to generate robust plans against unpredictable disturbances. Besides, the approach is expected to replan if disturbances occur while performing the plan. We propose this Decision Tree (DT)-based sequential approach (DTS) that solves the TUSP by sequentially making a sub-decision according to the DT prediction. It generates solutions that are both feasible and robust. Furthermore, it operates fast using the pre-trained model. We conduct experiments and compare its performance with a heuristic algorithm and the Local Search algorithm (LS). The proposed approach DTS solves fewer problems than LS and the heuristic, but it outperforms others by generating more robust solutions.

1. Introduction

The trains need to be cleaned and maintained at regular intervals to ensure a pleasant and safe journey for more than 1.1 million train passengers every day in the Netherlands [16]. Those cleaning and maintenance tasks are performed in service sites. Due to the limited space in service sites, a reasonable plan and schedule should be designed and performed satisfying all requirements and constraints. Currently, the plans and schedules are manually generated by experienced planners. The rolling stock is on service during the rush hours. Outside of those, only a small portion is needed for the reduced transportation demands. The surplus of the rolling stock park at service sites nearby major stations. Trains arrive in the evening and park at available tracks, then leave in the next morning. They will shunt to perform service tasks if there are some, and park at idle tracks after tasks are finished.

In this study, we tackle a limited version of TUSP containing parking, matching and routing subproblems. Note that parking a sequence of trains at tracks resembles the *Bin Packing Problem* which is well-known to be NP-hard

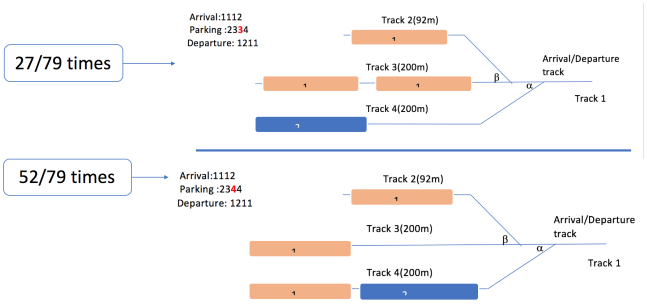


Figure 1: The randomness in the parking solutions to an example set of instances. All problem instances share the same arrival and departure sequence in the example set. An orange block is a train with material_id 1 and a blue one with materail_id 2. The sequence of "Arrival" and "Departure" is the list of trains represented by their material id. The sequence of "parking" is the parking assignment indicating the track id in the order of arrivals. The red "3" in "parking: 2334" means the third arrived train is parked at Track 3. In 27 out of 79 times LS generates the first parking solution and the rest 52 times it creates the second one.

[22]. Dutch Railways (Nederlandse Spoorwegen; NS) has conducted several researches and succeeded in producing feasible solutions to TUSP. However, the solutions have a lack of robustness (see an example in Figure 1). In practice, it is preferred to obtain robust solutions to shunting problems so that it can cope with unpredictable uncertainty(e.g., train arrival delays).

Shunting problems have been studied along 2 main directions. One is the standard optimization that requires complete input information about incoming and outgoing sequences of train units. Many proposed algorithms for TUSP succeed in finding feasible solutions by this method [10, 11, 12, 13, 14, 15, 21, 23]. However, these are not able to cope with disturbances in the input information that makes the original plan infeasible. The other algorithms model the shunting problem as an online planning problem to cope with arrival delays [9, 26]. It makes decisions on the basis of each event (e.g., arrival/departure) and reacts fast against disruptions.

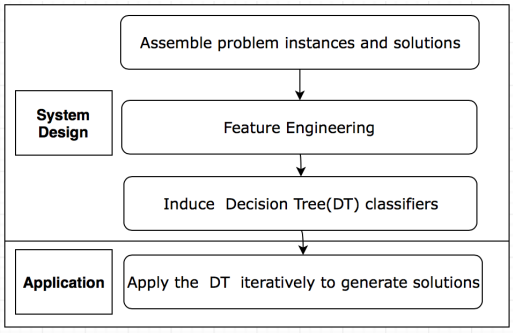


Figure 2: The system design and application algorithm

The research goal of this study is to investigate whether it is possible to generate robust solutions to TUSP using machine learning techniques. The fundamental task is to develop a solution approach that is capable of producing solutions both feasible and robust with interpretable machine learning techniques. We propose an approach that makes schedules using Decision Trees in a sequential manner. The proposed approach consists of 2 phases, design and application (see Figure 2). In the design phase, we collect training data and construct the DT models, which subsequently are applied for making schedules in the future period.

We employ the Decision Tree (DT) as the classification method because of several reasons. Firstly, it provides the direct insight into the principles and characters that lead to decisions. This property of interpretability is essential for human planners to understand its output. Secondly, a DT model is capable of distinguishing between multiple classes. The labels in the training set are the track ids. The number of classes is the number of tracks. At last, a DT model can take multiple types of parameters as input including categorical data, such as the material type of trains.

We evaluate the performance according to the number of instances it solves and the robustness of solutions. In robustness optimization, the robustness is measured as the trade-off between the performance of the robust solution and the performance of the optimal solution [3, 8]. To the best of our knowledge, there is not a standard quality measurement for a single feasible solution to TUSP. Therefore, we propose to count the number of unique solutions as a measurement of the robustness. A robust algorithm can generate feasible solutions that stay unchanged in every likely scene. In a fixed scenario, an algorithm is more robust if it solves problems with fewer unique solutions.

Our main contributions are three-fold:

1. We develop an approach that generates solutions sequentially for the train shunting problem in service sites. From experiments it demonstrates a high capability to produce solutions that are both feasible and robust.
2. We demonstrate how to use the Decision Tree (DT) classifier in predicting actions in each intermediate state and interpret the decision rules by visualizing the tree structure.
3. We provide a feature set that captures necessary information to make schedules by machine learning techniques.

The remaining paper is organized as follows. Related work is addressed in section 2, followed by the problem definition and the system design in section 3 and the proposed algorithm in section 4. The experiments settings and results are explained in section 5. In the end, we conclude and discuss the research project in section 6.

2. Related Work

In this section, we review related work in the train unit shunting problem, intelligent scheduling, robustness, and Decision Tree.

2.1. Train Unit Shunting Problem

The Train Unit Shunting Problem (TUSP) is first introduced by Freling et al.[10] and it consists of 5 subproblems, namely matching, parking, service scheduling, routing and crew planning. In some researches, they make a decomposition of the complete problem and solve subproblems sequentially [10, 15, 14, 23]. In contrast, there are also several integrated approaches that solve the whole problem together [13, 21, 12, 11].

In a sequential approach, the matching subproblem is solved first by modeling as a Mixed Integer Programming problem in [10, 15, 14]. Then a parking plan is generated by a column generation approach [10]. Afterward, one extension is to solve the routing subproblem by estimating the routing cost through a graph representation of the shunting yard [15]. It reduces the computation time for generating acceptable solutions through decomposition, but it also causes the loss of global optimality.

As for the integrated approach to the combined matching and parking subproblems, one can group the massive amount of crossing constraints in clique constraints in the mathematical formulation [13]. A second approach is an exact dynamic programming algorithm taking the solutions from a greedy heuristic [21]. Besides, the combined subproblems can also be solved by genetic algorithms [12] or heuristics [11]. It takes a longer computation time for integrated approaches, from 10 minutes up to hours, due to the increased problem complexity.

The best approach so far in NS is an integrated Local Search approach presented by Van den Broek which solves the 4 subproblems for instances with 23 trains composed

of 27 train units in 4 minutes [22]. It is a simulated annealing algorithm that starts with an initial plan. Then, it explores through the search space by evaluating each subproblem and simultaneously constructs the shunting plan. The algorithm starts with a random initial solution every time we run it. After iterations, it leads to different final solutions. Because of the assumption, that the service site is empty and no train is parking at the beginning of planning, it can not replan after trains arrival. Thus it fails when a small delay occurs resulting in the original plan to be infeasible during the plan operation.

2.2. Intelligent Scheduling

For a scheduling problem, one method is static scheduling which generates a detailed operations plan at the beginning of the planning horizon in order to achieve global optimization [27]. Several scheduling approaches apply this method in the train shunting problem [22, 11, 6, 10, 12, 13, 14, 15, 21, 23]. This standard optimization approach needs perfect knowledge about the arrival, departure timetables and the all relevant information about the train units for scheduling in train shunting yards.

The dynamic dispatching means that we generate one next action in each time when an operation process is finished and some candidate operations are available [27]. This approach is usually applied in real time planning cases, for example in online dispatching rule selection in the manufacturing system [19], container transportation planning to inland services [24] and online dispatching of train units or trams [9, 26]. Real-time scheduling does not require complete knowledge for specific planning horizon like standard optimization approaches. It can generate plans dynamically under uncertainties.

The proposed approach takes all given information as input to achieve the feasibility in the solution. It also follows the sequential scheduling manner as in online planning to react fast against uncertainty.

2.3. Robustness

The robustness has long been a focus of the scheduling problem, but a general definition of the robustness is still missing. Informally speaking, the robustness is the attribute of the schedules to remain insensitive against disturbances during operations [6]. There are 2 main focuses of robust models, robustness optimization (RO) and stochastic programming. In RO, one aims to find solutions that are feasible for any realization of uncertainty in a given set in which the disruptions are determined and set-based. Bertsimas et al. [2] give several examples of RO to obtain solutions with different desired properties, like sparsity, stability, and statistical consistency. Cicerone et al. propose a recoverable robust model for the shunting problem and measure the robust solutions by the price of robustness [3] that determines

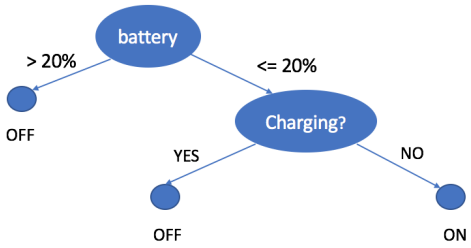


Figure 3: An example of the Decision Tree application showing the decision process of turning on or off of the battery saving mode. If the input is associated with 'battery power' less than 20% and not charging, the tree will navigate it to an 'ON' decision by following the rightmost path.

the tradeoff between an optimal solution and a robust one [7, 8]. Entropy is also applied as a measurement of the consistency in solutions [17]. Stochastic programming requires the probability distribution of the disruption scenarios (see Birge and Louveaux 2011 [4]) which is not trivial to obtain in practice. In this project, we aim to react fast and keep robustness in plans when arrival delays occur during operation. Thus, we would not rely on the probability distribution of the disturbances but restrict in a fixed set of scenarios.

2.4. Decision Tree

A Decision Tree classifier is a method applied to construct complex decision-making. It is expressed as a rooted tree that can recursively partition among the instance space which is the set of all possible examples. It represents the general relationship between the input attributes and target attributes [18] (see an example in Figure 3).

The Decision Tree model is applied dynamically to choose the appropriate dispatching rule given a set of system attributes in the manufacturing system [19]. Then a hybrid model delivers better generalization ability that employs an artificial neural network to identify the significant system attributes [20]. DT is also applied to solve the optimal ordering problem in sequential auctions to obtain good orderings with high revenues [25].

For the train shunting problem, the search space is vast as the number of choices of candidate trains and actions is rather large. This amount grows as more and more trains involved, which makes it challenging to formulate the problem and generate plans.

2.5. Summary

To summarize, the approach to generate robust solutions to TUSP is missing. We propose to design an algorithm in a sequential manner like dynamic dispatching to cope with uncertainty. Machine learning techniques are able to find the common patterns and rules from historical data to make

predictions so that it leads to robust solutions. Especially, the Decision Tree algorithm, as a white box method, is capable of processing multiclasses data and categorical features.

3. System Design

In this section, we provide a formal problem definition of TUSP in our study followed by the introduction of our system design. The system design consists of 3 steps as shown in Figure 4. Firstly, we assemble training data from the simulation system provided by NS. An instance generator generates real-life scenarios of arrivals and departures with a timetable and train material types. Then we employ the Local Search Algorithm [22] to generate solutions for problem instances. After the assembly of instances and solutions, we process the feature engineering to prepare the training data for DT classifiers, details in subsection 3.2. Subsequently, DT classifiers are inducted by taking the training data as input from step 2, details in subsection 3.3.

3.1. Problem Definition

In this research, a fictional shunting yard is designed in our experiments which is in a 'shuffleboard' structure. TR denotes the parking tracks set in which there are r First-In-Last-Out tracks each with a length tl. There is one $gate\ track$ connected to all r parking tracks. We define a set of arrival trains AT, in which each train at_i is associated with an arrival time at_i^{tm}, a material type at_i^{tp}, a train length at_i^l and a parking track at_i^p $(i = 1, .., n)$. There is a departures set DT in which each departure dt is assigned with a departure time dt_i^{tm} and a material type dt_i^{tp} $(i = 1, ..., n)$. We assume each train consists of one train unit. Then the train material type equals to the train unit subtype, e.g., ICM-4 represents a train that consists of 4 carriages of material ICM (= Intercity Material).

Given AT, DT and TR, we aim to find (1) a parking assignment for all arrival trains to parking locations; (2) a matching assignment for all parking trains to departure positions such that

$$\sum_{j=1}^{n} at_j^l \leq tl_i \text{ if } at_j^p = tr_i \text{ for } i = 1, ..., r \qquad (1)$$

, the sum of parking trains length in one track is no larger than the track length,

$$at^{tp} = dt_j^{tp} \text{ for } j = 1, ..., n \qquad (2)$$

, the material type matches for each departure.

For each arrival, a specific train unit is assigned in the input. For each departure, it only requires the train material type but not a physical train unit. There is flexibility to choose from several trains as long as the material type matchs with the departure requirement.

3.2. Feature Engineering

Intuitively, one may predict a complete plan for each problem instance. However, machine learning models are not capable of predicting among the enormous amount of unique solutions. Thus, we propose this DT based sequential approach which makes predictions for each event to reduce the prediction space. Each arrival and departure is taken as an event for which the model predicts an action that leads to feasible and consistent solutions. For each event, we collect the current track occupation, future events, and the trigger information, which we call a *State*, as input features to the DT classifier. It will output a vector of probabilities of each *Action*. One main advantage of this (State, Action) modeling is that the model can start planning at any time point by collecting the current State.

State: There are 3 components of *State* attributes, the track occupation, future events and trigger information. The first one provides detailed information about whether a parking position is occupied, and if true by which train material type. Besides, we also calculate several aggregated features to give a direct insight into the current situation. It is necessary to consider the track occupation to avoid the track lengths violation due to the limited track length. The future events part presents the information about the material type of next arrivals and departures. The model should take it into account so that the parking assignment will not cause obstructions while departing. The trigger part describes necessary details of the current event.

Track Occupation: Based on our observation of historical solutions, the parking capacity for a track is 3 trains. Therefore, we define $3r$ parking position features in a shunting yard with r parking tracks. Its value is the material type id of the train that is parking at that track position. For example, "track_1_position_0" with value equals to "1" means a train with material type id 1 is parking in track 1 at position 0. Besides, there are two aggregated features describing the track occupation. "parking_trainunits" provides the number of parking train units in the shunting yard, and "empty_tracks" records the number of unoccupied tracks.

Future Events: We consider the next n events of arrivals and departures respectively, and it is $2n$ features in total. The value of a next arrival is the material type id of that arrival train. For a next departure, its value is the required material type id of that departure. There is an additional aggregated feature, "arriving_trainunits", giving the exact number of future arriving train units.

Trigger Information: "trigger_type" indicates the event type of the current one, e.g. 1 = a departure and 2 = an arrival. "trigger_material" shows the material type id that is associated to this event. "state_id" records an integer number showing the rate of progress in solving the problem.

Action: We define the target set H as the set of all parking tracks $(tr_1, ..., tr_r)$ in the shunting yard. An action is

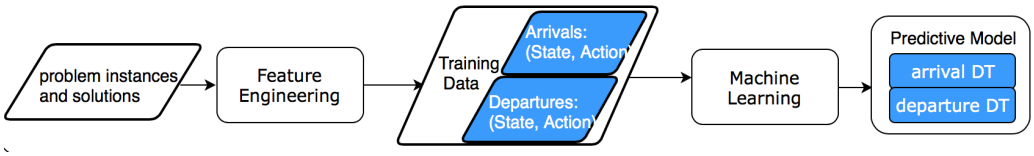


Figure 4: System design process. We collect a set of problem instances and their solutions, which subsequently are used to generate 2 training sets by feature engineering. Then, we construct 2 predictive models from those 2 training sets.

defined as "move a train from a start track tr_A to an end track tr_B". For each event, since we know the associated train, the DT classifier only outputs a vector of the probability distribution over *r* tracks. If the event is an arrival, we select one parking track from the set H, as the end track tr_B in the action. For a departure event, a train will leave from the selected track tr_A to the $gate\ track$. When there are several trains parking at the selected track, the one parking at the front will be chosen to depart.

3.3. Decision Tree Inference

In order to obtain a model that performs well in all circumstances, we construct an individual DT classifier for arrivals and departures respectively. The feature sets of two events are identical. However, the value of some features distributes with a significant difference. For example, as we assume that all trains arrive before the first departure, the value of the next arrivals features is always *None* in departure events but not *None* in arrival events. A difference like this may cause additional misclassification. Therefore, we separate two events and construct two different classifiers.

We utilize the CART algorithm [5] as the DT inference method with Gini splitting criterion. It partitions the instance space recursively in each node by selecting the feature that gives the least impurity in child nodes according to the Gini Index [18]. The tree inference continues partitioning until a stopping criterion fulfilled: in this case, we set it as the maximum tree depth d. According to Breiman [5], the tree complexity shows a crucial influence on its performance. We choose the maximum tree depth and the number of nodes to measure the tree complexity. The final d will be determined considering the performance of solving problem instances.

For each leaf node, it is associated with a vector of class distribution of all observations classified in that node. If all observations belong to a single class and Gini Index equals to 0, the leaf will be assigned to that class. If the Gini Index is positive indicating some impurity, the leaf node will be designated to the largest class.

4. DT Based Sequential algorithm(DTS) in TUSP

In this section, we introduce our proposed algorithm **DTS** and one variant **DTS_1**. The categorical features are encoded as integers in DTS, and we transform them by one-hot encoding in DTS_1.

Algorithm 1 DTS

procedure BUILD SOLUTION($AT, DT, TR, gate\ track$)
  Sort arrivals AT by time desc
  Sort departures DT by time desc
  Initialize current state s
  Initialize an action list A
  while AT and DT not empty **do**
    if current event is an arrival **then**
      predict track probabilities P_r by $arrival\ DT$
      target tracks $H \leftarrow TR$ sorted by P_r
      $tr_B \leftarrow$pop the track with max P_r from H
      $tr_A \leftarrow gate\ track$
      while $Action(tr_A, tr_B)$ not valid and H not empty **do**
        $tr_B \leftarrow$pop the track with max P_r from H
    else if current event is a departure **then**
      predict track probabilities P_r by $departure\ DT$
      target tracks $H \leftarrow TR$ sorted by P_r
      $tr_A \leftarrow$pop the track with max P_r from H
      $tr_B \leftarrow gate\ track$
      while $Action(tr_A, tr_B)$ not valid and H not empty **do**
        $tr_A \leftarrow$pop the track with max P_r from H
    if $Action(tr_A, tr_B)$ not valid **then**
      break
    else
      add $Action(tr_A, tr_B)$ to A
    Compute s', AT', DT'
  return the action list A

The algorithm starts from building an initial state from the problem state as shown in Figure 5. It utilizes the DT classifiers on a per event basis and generates the solution sequentially(see algorithm 1). Firstly, the algorithm assembles the current problem State s and identifies the event type. The arrival state features will be sent to the $arrival\ DT$ whose output is a vector of possibilities of each track. The selection of tracks follows a greedy manner. The track with the highest probability will be chosen first as the parking track. Then it forms an action a of moving the arrival train from the $gate\ track$ to the predicted parking track tr_B. If the action a is not valid in this state s, a second track will be selected until the action is valid or all

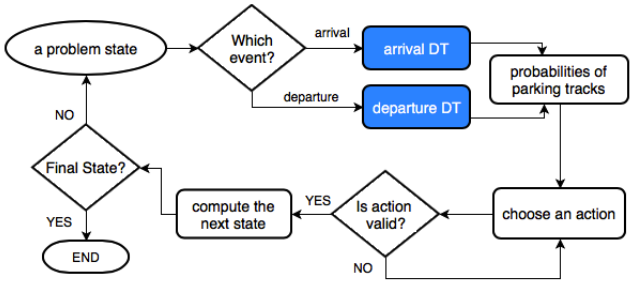


Figure 5: DTS working flow chart. For a problem instance, the DTS forms an initial state as the start. The state will be sent to the corresponding DT model according to the event type, and the model predicts a list of probabilities for different actions. DTS selects the action with the highest probability and checks if the action is valid. It transfers to the next state if the action is valid, or choose another one if not.

possible parking tracks are checked. For an arrival event, we check whether the remaining tracks length is enough to park the arrival train. Likewise arrivals, a departure state will be classified by the $departure\ DT$ whose output is the probability distribution of all tracks. The action for departure events is defined as "move the front train from the predicted track tr_A to the $gate\ track$". For a departure, the model will check whether the train material type matches with the request. We obtain a feasible solution if all actions are valid in that state and the construction ends in a final state with no remaining arrivals or departures. The algorithm fails if it can not find any valid action in a state during the construction.

DTS with Onehot Encoded Features: There are several categorical features in our feature set, e.g., the next arrivals whose value is the train material type, "SLT-4". In our basic version of implementation, we encode those features as integers, which means each category is represented by an integer. Decision Tree is capable of processing integer encoded categorical features. However, it will assume there are ordinal relationships existing and it might result in poor performance or unexpected outcomes. Therefore, we implement another version of the proposed approach with the categorical features transformed by one-hot encoding, shortened as **DTS_1** in the results tables. For a feature with m categorical values, a binary variable is introduced for each unique category, and the old categorical feature is removed. We avoid the assumption of natural ordering between integer values, and it also improves the interpretability of the Decision Tree models. A shortcoming is that the increased number of features may result in overfitting.

5. Experiments

In this section, we conduct extensive experiments by applying DT based sequential approach on an elaborate scenario. We use the solutions generated by the Local Search algorithm [22] in the training data. Then we compare the performance of the Local Seach algorithm, a heuristic method, **DTS**, and **DTS_1**. The comparison results are analyzed in subsection 5.5.

5.1. Scenario

In this scenario, we design 2 subsets of 200 instances of medium and difficult according to the difficulty level of solving the problems. For a problem instance of medium level, the number of trains is slightly larger than the number of tracks. In a few tracks, there are 2 or more trains parking that results in certain flexibility in solutions. That is the main scenario that our model focuses on. In difficult problem instances, there are more trains and the usage of track length is rather high. For each instance, there are only a few solutions with limited flexibility.

We design a shunting yard with $r = 9$ FILO tracks and one gate track according to the shunting yard layout in Kleine Binckorst [22]. There are 6 different material types whose train lengths and the arrival ratio are settled regarding the daily timetable in Kleine Binckhorst (see Table 1). We define two subsets of the problem instances, one with $n = 10$ train units and the other with $n = 12$. The set of trains material type of instance with problem size 10 is (2 x SLT-4, 2 x SLT-6, 2 x VIRM-4, VIRM-6, 2 x ICM-3, ICM-4). The track usage rate for 10 and 12 train units is 56% and 65% respectively. For this layout, a track can park at most 2 - 3 train units, and there are 23 parking positions in total. For the future events, we provide the next $k = n$ arrivals and departures individually. The target set is the set of all parking tracks $(tr_1, ..., tr_9)$. To induct the DT classifiers, we generate 2k problem instances and solutions for each subset. Then we obtain a training set of 20k to 24k samples for each DT classifier. This set is split into a training set and a test set by 8:2 and the prediction accuracy is averaged after cross-validation.

5.2. Evaluation

The goal of this study is to generate robust and feasible solutions to TUSP. Therefore, we evaluate the model performance according to the number of instances solved (NF), and the robustness in the solutions. The parking assignment affects the departures assignment significantly. In contrast, the departure events show less influence on further planning. Therefore, the robustness evaluation focuses on the parking assignment that is called "solution" in the remaining section. We count the number of unique solutions NU to all solved instances in the test set. The parking solution is

Table 1: Train unit information. The arriving ratio shows the appearance probability of a train with that material type in one instance.

material type	train length (m)	arrival ratio
SLT-4	70	0.21
SLT-6	101	0.25
VIRM-4	109	0.18
VIRM-6	162	0.05
ICM-3	81	0.21
ICM-4	107	0.10

a sequence of end tracks that the arrival trains are assigned to park. Two solutions are considered the same if two sequences are identical. It measures the model's ability to generate robust solutions for instances with the same problem size. The fewer number of unique solutions, the more robust. Besides, we also compute the ratio between NF and NU to reflect the robustness in the solutions shorted as RUF. The smaller RUF, the more robust. Overall, the objective of the algorithm is to solve more problems with fewer unique solutions.

5.3. Baseline Algorithms

In order to present a better understanding of the performance of the proposed approach, we make a comparison among the Local Search algorithm [22], a heuristic algorithm and 2 versions of the proposed algorithm. The Heuristic works as the benchmark of number of instances solved. It may not solve all instances, but the solutions would be robust. The LS algorithm is expected to solve the most amount of instances but the solutions are rather random.

Heuristic: This heuristic algorithm is adjusted based on the solution method developed by Beerthuizen [1]. It follows a residence time strategy and acts as following rules. For parking assignment, each arrival train is labeled with a priority index according to the departure request , e.g.1 = the first to depart. Each arrival train is associated with the first departure with the same material type. When a train arrives, it will be parked at a track where the train parking at back has a larger priority index and the least difference in priority. Otherwise, it will park the train in an empty track.

DTS+LS: The proposed algorithm is expected to solve problem with few unique solutions. However, it may show poor performance in terms of the number of solved problems when the instance turns difficult. Therefore, we propose to combine DTS with LS to achieve a better performance. For a problem instance, we first run DTS. If no feasible solution found, we use LS to solve the problem.

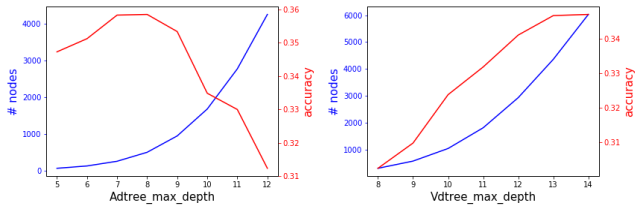


Figure 6: Comparison of Decision Trees for arrival events(left) and departure events(right) with different max tree depths in instances subset with 12 train units.

5.4. Parametric Exploration

To obtain appropriate DT models that lead to robust and feasible solutions, the maximum tree depth needed to be determined. With the instances subset of 12 train units, we explore through max tree depth d_A in the range of (5,12) for the $arrival\ DT$ and d_V in the range of (8,14) for the $departure\ DT$. From Figure 6, we notice the prediction accuracy of the $arrival\ DT$ rises till $d_A = 8$, and drops afterward, indicating possible overfitting when $d_A > 9$. For the $departure\ DT$, the prediction accuracy increases as the tree depth d_V arises. As a result, we reset the parameters range as $d_A \in (5, ..., 9)$ and $d_V \in (8, ..., 14)$ in the further exploration.

We investigate the impact of max tree depth on the algorithm's performance regarding the number of instances solved (NF), the number of unique solutions (NU), and the ratio of unique solutions to feasible solutions (RUF) (results in Figure 7). We test it on a validation set of 200 instances with 12 train units. In each row, for a fixed d_A, NF reaches the peak when $d_V = 8$ or 9 (the highest blue bar). NU does not vary obviously, so the lowest RUF locates at $d_V = 8$ or 9 (highlighted in orange squares and the RUF value in red). For a fixed d_V (e.g., $d_V = 8$), NF fluctuates for different d_A, but NU arises dramatically as d_A increases (see the green bar rises in a column). RUF shows a huge jump from the top row to bottom.

To sum up, we select $(d_A = 5, d_V = 8)$ to train the DT models in DTS for the instances subset with 12 train units. A same process applies to other subsets and the combination $(d_A = 5, d_V = 8)$ is chosen for all of them .

5.5. Results

We compare the performance of 4 algorithms concerning the number of instances solved, the number of unique solutions, the ratio of unique solutions to feasible solutions, and the average runtime (results in Table 2). The model performs best if it solves more problem instances, with fewer unique solutions. For problems with 10 train units, DTS and DTS_1 solve around 112 instances with only 3 unique solutions. Heuristic solves more problems with more unique

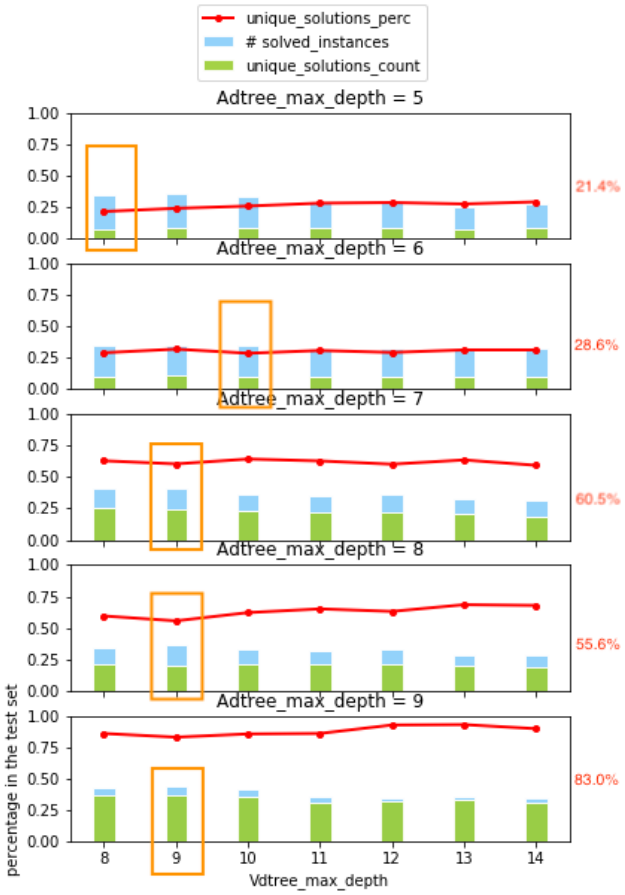


Figure 7: Comparing the performance of DTS with different max tree depth in the instances subset with 12 train units. For trees with a larger max depth, the model can solve more problems as the blue bar raises. But the number of unique solutions also increases with trees with a larger max depth as the green bar grows. For a fixed arrival DT max depth, the best corresponding departure DT max depth is highlighted by the yellow rectangle. The ratio of unique solutions to feasible solutions rises with tree grows indicating the decrease of robustness in solutions.

solutions. And LS solves all instances with 200 different solutions. The best one is LS+DTS, which solves all 200 problems with 91 unique solutions. For problems with 12 train units, DTS and DTS_1 generate the fewest amount of unique solutions, but DTS solves slightly more instances than DTS_1. Heuristic solves 2 times as many instances as DTS but with 3 times as many unique solutions as DTS. LS provides a unique feasible solution to each of the 200 instances. The best one is still DTS+LS which solves all instances with 153 unique solutions. The run time of different

Table 2: Performance comparison of algorithms in test set of 10 train units and 12 train units. The ratio is between the number of unique solutions and the number of feasible solutions. "H" represents the "Heuristic".

Alg		DTS	DTS_1	H	LS	DTS+LS
10 train units	solved	112	114	187	200	200
	unique	3	4	157	200	91
	ratio (%)	2.67	3.51	84.0	100.0	45.5
	runtime (s)	0.94	1.28	1.32	1.77	1.72
12 train units	solved	61	57	134	200	200
	unique	14	13	133	200	153
	ratio (%)	23.0	22.8	99.3	100.0	76.5
	runtime (s)	0.96	1.17	1.37	2.48	2.68

algorithms does not differ significantly in this test set. DTS and DTS_1 are rather time efficient than other algorithms.

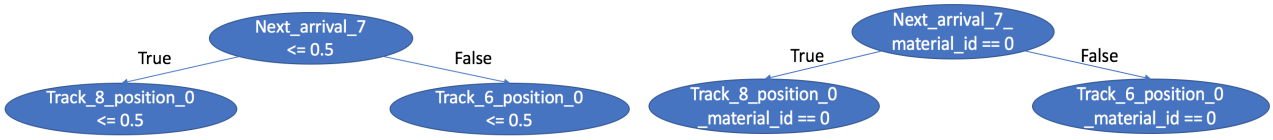
Interpretability of DT models: We visualize the inferred $arrival\ DT$ applied in DTS and DTS_1 in Figure 8. The top structure of 2 trees demonstrates a same logic but in slightly different representations. The tree from one-hot encoded features is more comprehensive since it avoids the assumption of natural ordering in integer values.

6. Conclusion and Discussion

In this study, we propose a DT based sequential approach to generate robust and feasible solutions to TUSP. In the design phase, we collect training data generated from the instance generator and Local Search algorithm in NS. Then, we build DTS and DTS_1 algorithms with pre-trained DT models. In the experiments, two proposed algorithms outperform the Heuristic and LS regarding the number of unique solutions that indicates strong robustness. There is no apparent difference in the performance of DTS and DTS_1 showing that the encoding method has little effect in this case.

The proposed DTS and DTS_1 solve relatively fewer instances than other methods. One reason is that there exists randomness in the training data from LS. Then the labels attached to the samples are not guaranteed to be the ground truth in the training set. DT infers the decision rules considering those labels which make it difficult to induce an accurate DT model. The second reason is that errors accumulation during sequential scheduling. The algorithm's failure in one state may result from an inappropriate action in previous states, and there is no mechanism to correct it. The third reason could be that we make a tradeoff between solving more instances and generating fewer unique solutions during DT models construction. A small DT model loses some details to plan less common cases, but it leads to robust solutions.

Considering the limitation in solving more instances, firstly we suggest to collect less random solutions as training data to the DT model. For example, assemble the solu-



(a) arrival DT in DTS. **(b)** arrival DT in DTS_1 (onehot encoded features).

Figure 8: arrival DT top structure for the training set with 12 train units (tree depth = 5). In the tree at left, the root node checks the value of feature 'next_arrival_7'. If it is no larger than 0.5 (means there is no 7th arrival train), then it goes to the left child node. In the tree at right, the root nodes checks whether the next_arrival_7_material_id equals 0 (means there is no 7th arrival train). If true, then it goes to left child node to check whether there is a train parking at track_8_position_0.

tions generated by DTS to reduce the noise in the training set. The second suggestion is allowing temporary relocation during planning. We could cope with the obstruction while departing or track length violation during arrivals by enabling a train to relocate at another available track in the shunting yard. An alternative way is to allow the algorithm to move few states back and select a different action if it fails to find a valid action at the current state. Thirdly, we can create initial solutions by DTS to LS since the randomness comes from the arbitrarily selected start point. LS may produce robust solutions by starting with a robust one.

References

[1] E. Beerthuizen. Optimizing train parking and shunting at ns service yards. Master's thesis, 2018. 7

[2] D. Bertsimas, D. B. Brown, and C. Caramanis. Theory and applications of robust optimization. *SIAM review*, 53(3):464–501, 2011. 3

[3] D. Bertsimas and M. Sim. The price of robustness. *Operations research*, 52(1):35–53, 2004. 2, 3

[4] J. R. Birge and F. Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011. 3

[5] L. Breiman. *Classification and regression trees*. Routledge, 2017. 5

[6] V. Cacchiani, A. Caprara, L. Galli, L. Kroon, G. Maróti, and P. Toth. Railway rolling stock planning: Robustness against large disruptions. *Transportation Science*, 46(2):217–232, 2012. 3

[7] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. 12. robust algorithms and price of robustness in shunting problems. In *OASIcs-OpenAccess Series in Informatics*, volume 7. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007. 3

[8] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. Recoverable robustness for train shunting problems. *Algorithmic Operations Research*, 4(2):102–116, 2009. 2, 3

[9] M. Demange, G. Di Stefano, and B. Leroy-Beaulieu. On the online track assignment problem. *Discrete Applied Mathematics*, 160(7-8):1072–1093, 2012. 1, 3

[10] R. Freling, R. M. Lentink, L. G. Kroon, and D. Huisman. Shunting of passenger train units in a railway station. *Transportation Science*, 39(2):261–272, 2005. 1, 2, 3

[11] P. M. Jacobsen and D. Pisinger. Train shunting at a workshop area. *Flexible services and manufacturing journal*, 23(2):156–180, 2011. 1, 2, 3

[12] D. Jekkers, I. U. Kaymak, and L. G. Kroon. Train shunt planning using genetic algorithms. Master's thesis, 2009. 1, 2, 3

[13] L. G. Kroon, R. M. Lentink, and A. Schrijver. Shunting of passenger train units: an integrated approach. *Transportation Science*, 42(4):436–449, 2008. 1, 2, 3

[14] R. Lentink. *Algorithmic decision support for shunt planning*. Number 73. 2006. 1, 2, 3

[15] 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*, pages 415–436, 2006. 1, 2, 3

[16] NS. Passenger rail service. 1

[17] E. Peer. Shunting trains with deep reinforcement learning. Master's thesis, 2018. 3

[18] L. Rokach and O. Maimon. Top-down induction of decision trees classifiers-a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(4):476–487, 2005. 3, 5

[19] M. J. Shaw, S. Park, and N. Raman. Intelligent scheduling with machine learning capabilities: the induction of scheduling knowledge. *IIE transactions*, 24(2):156–168, 1992. 3

[20] Y.-R. Shiue and C.-T. Su. An enhanced knowledge representation for decision-tree based learning adaptive scheduling. *International Journal of Computer Integrated Manufacturing*, 16(1):48–60, 2003. 3

[21] 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. 2008. 1, 2, 3

[22] R. W. van den Broek. Train shunting and service scheduling: an integrated local search approach. Master's thesis, 2016. 1, 3, 4, 6, 7

[23] M. T. van Dommelen. Scheduling train service activities at service sites to determine the capacity. Master's thesis, 2015. 1, 2, 3

[24] B. Van Riessen, R. R. Negenborn, and R. Dekker. Real-time container transport planning with decision trees based on offline obtained optimal solutions. *Decision Support Systems*, 89:1–16, 2016. 3

[25] S. Verwer, Y. Zhang, and Q. C. Ye. Auction optimization using regression trees and linear models as integer programs. *Artificial Intelligence*, 244:368–395, 2017. 3

[26] T. Winter and U. T. Zimmermann. Real-time dispatch of trams in storage yards. *Annals of Operations Research*, 96(1-4):287–315, 2000. 1, 3

[27] S. D. Wu, E.-S. Byeon, and R. H. Storer. A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, 47(1):113–124, 1999. 3

2

Introduction

This research is operated as my master thesis project during my internship in NS Techniek, that is a subsidiary of Nederlandse Spoorwegen (`Dutch Railways; NS`), the largest railway operator in the Netherlands. NS Techniek is in charge of the maintenance, cleaning, and refurbishment of rolling stock in NS. Currently, the plans and schedules are manually generated by experienced planners. It is not a trivial task even for experienced planners to generate a schedule. It is becoming more tedious and time-consuming because of the increasing rolling stock.

We focus on the Train Unit Shunting problem(TUSP) in the service site in this study. Given the layout of the service site, the timetable of the trains' arrival and departure, and the service tasks needed to perform, the planners generate a schedule for trains in a 24-hour horizon. A complete schedule contains the assignment of trains parking locations, the start time and the location of each service task, the matching of the departure positions, and a shunting plan of all movements. There are several constraints when constructing a schedule. For example, a train can only park at a track if its length is shorter than the track useful length. The service tasks can only perform at specific tracks that are equipped with required facilities. There are also requirements for each departure about the train units material type and the train composition.

Some researches have succeeded to find feasible solutions to TUSP. However, those approaches either fall short in lack of consideration of real-life situations or take hours of computation time. The best approach in NS is the Local Search algorithm designed by Roel van den Broek [1] that can generate a solution for easy problems within 2 minutes. The computation time rises to 10 minutes when the problem becomes more complicated. The existing approaches mainly focus on solving problems in a short computation time. The robustness of solutions is still missing. One property we want to obtain in the approach is to replan. When disturbances occur during the plan operation and the plan becomes infeasible, we generate a new plan from the failing point. Another property is the ability of remaining robustness against a low level of disruption(e.g. arrival delays). In other words, when there is a small disturbance, we would like to make little adjustments to the original plan.

2.1. Randomness in the Solutions

The Local Search approach searches through the solution spaces by starting from a random initial start point in each iteration. So, when there are some delays while execution and the original solution becomes unfeasible, we need to rerun the algorithm and it may generate a new schedule with radical changes compared to the original one. One example is visualized to give a thorough perception of the randomness. First, we introduce the problem settings and assumptions for one problem instance, then explain the randomness through visualizations of the solutions.

2.1.1. Problem Description

A problem instance consists of several components collectively providing complete information that needs to consider in scheduling. We introduce each segment through an example in the following section. In the end, there is an example solution for the example problem instance.

Service Site Layout:

In this example, there are three tracks for parking and one gate track for arrivals and departures as in figure 2.1. Trains arrive from the gate track in the evening by a certain order and park at one of those tracks. All of them leave by departure requirements in the next morning. In this case, only the right sides of parking tracks are open sides. Trains can only shunt to parking tracks from the right side of tracks following a rule of First In Last Out.

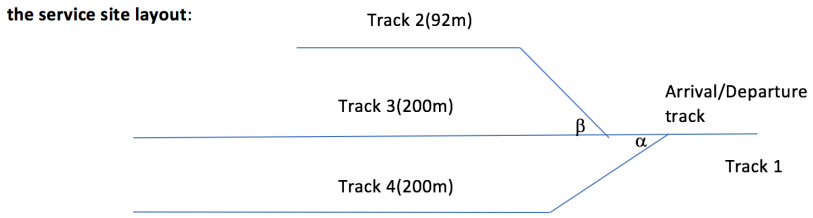


Figure 2.1: The service site layout with 3 parking tracks (Track 2/3/4) and 1 gate track (Track 1). Trains can not park at the gate track. The number following the track id is its useful length.

Train and train unit:

In the example problem instance, there are four trains and each of them consists of one single train unit. There are 2 types of train unit with different material type and lengths. More details are given in the table 2.1.

Table 2.1: The details of the trains in the example.

material id	material type	length	amount in one instance
1	ICM-3	82m	3
2	ICM-4	107m	1

Timetable:

For a problem instance, there is an arrival timetable indicating the trains' id, arrival time, material type and train composition, etc. For departures, there are departures time, their material types and trains composition. It only requires the material type and train composition, but not any physical train unit in departures. So there is a certain level of flexibility to assign a train to a departure position as long as the requirements satisfied. Here are example timetables for arrivals 2.2 and departures 2.3.

Table 2.2: An example timetable (arrivals).

train_id	train_unit_id	material id	material type	length	arrival time
2000	4002	1	ICM-3	82m	20:45
1000	4001	1	ICM-3	82m	20:51
4000	4201	2	ICM-4	107m	20:54
3000	4003	1	ICM-3	82m	21:12

Solution:

A schedule for this problem is a sequence of movements indicating which train moves to which track. Given the information above, an example solution generated by the local search approach is provided in table 2.4. It shows a detailed schedule including the exact time point and location for each movement. In this case, there are eight movements in total, in which the first four related to parking.

Table 2.3: An example timetable (departures). In column "train composition (by material id)", a "1" means it requires a train composed of a train unit with materail id 1. In column "departure time", "+1" means it is the time on the second day.

train_id	train composition (by material id)	departure time
52000	1	5:23 +1
51000	1	5:44 +1
53000	1	5:47 +1
54000	2	6:03 +1

Table 2.4: An example solution for the example problem instance.The "+1" means the time in the next morning in column "start time" and "end time".

train_id	start time	end time	start track	end track
2000	20:45	20:48	1	2
1000	20:51	20:54	1	3
4000	20:54	20:57	1	4
3000	21:12	21:15	1	4
1000	5:20 +1	5:23 +1	3	1
2000	5:41 +1	5:44 +1	2	1
3000	5:44 +1	5:47 +1	4	1
4000	6:00 +1	6:03 +1	4	1

2.1.2. Randomness in Solutions

We have a set of 79 problem instances, and all of them share the same settings and assumptions. The example problem instance explained above is one of them, and the remaining 78 only differ in arrivals and departures time. The arrivals and departures sequence remains the same according to material type and train composition. The arrivals and departures of all instances are visualized in the scatter plot 2.2.

Each dot in the figure represents a trigger for an event (arrival/departure), and the color indicates the material type of the train unit. On the X-axis it records the time point for the trigger in minutes (18:00 equals to time 0). On the Y-axis it displays the instance id, in other words, triggers belong to one instance share the same value on the Y-axis. There is an obvious gap between time 500 - 650 indicating the last arrival happens before time 500 and the first departure after time 650. If we choose one as an original instance, the remaining ones are the deviations with disturbances in time. The instances remain comparable from a human perspective.

We run the local search algorithm for all instances under the same settings. It generates two different parking assignments with a fraction of 27:52 given in the figure 2.3. The difference happens when the third train arrives, either parking it in Track 3 or Track 4. This distinction leads to a significant difference in the overall schedules. At the time of the third arrival, the track occupation and remaining events in the timetable are relatively comparable in all instances. From a human aspect, one prefers to make the same decision and keep the consistency among solutions. Therefore, we propose a scheme of sequential scheduling for purpose of global consistency by achieving it locally.

2.2. Research Questions

In this section, we propose and discuss the research questions.

Can Decision Trees improve the robustness in solutions to the train unit shunting problem?

Instead of formulating the TUSP explicitly, we propose to utilize a training set of problem instances and solutions, then translate the planning rules automatically. It is not practical to construct an expert system for TUSP by interviewing human planners. A complete system requires a large number of rules

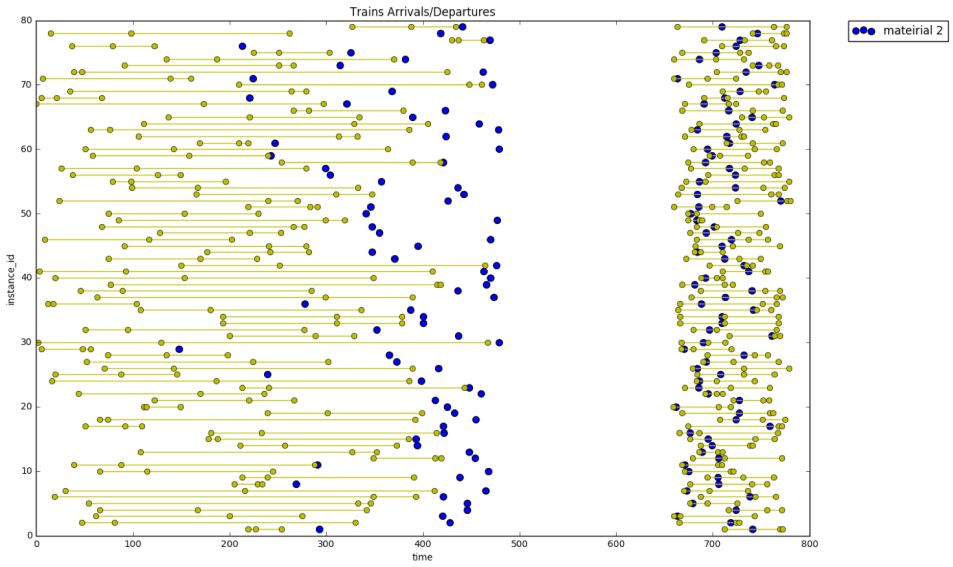


Figure 2.2: Timetable visualization for instances in the example set. A yellow dot is a train with material_id 1 and a blue dot is with material_id 2.On the X-axis it records the time point for the trigger in minutes (18:00 equals to time 0). On the Y-axis it displays the instance id.

in order to generate feasible solutions. As a result, the inference of all rules is expensive and time-consuming. Thus, we use the machine learning technique to automate the rule induction phase. The characteristics of this scheduling problem do not change over time, giving rise to periodicity, e.g., weekly timetable. Therefore, analytics on historical information can be used to find patterns and construct a machine learning model. More specifically, we would like to utilize the Decision Tree among all machine learning models because we can interpret the decision rules from the Decision Tree explicitly.

The solution space for train shunting problem is enormous, and it is not likely to make predictions efficiently using Decision Trees. Therefore, we propose to generate solutions sequentially by predicting one action from the current state in each step. By sequential scheduling, it reduces the prediction space and improves the prediction accuracy of the machine learning model. When a new event occurs (e.g., an arrival), the machine learning model can predict an action using the precepts learned from training data. This model needs little computation time after pre-training and can start at any time point in planning. Consequently, it can also react fast to disturbances such as arrival delays.

Subquestion: 1. How to represent a problem state for train unit shunting problem?

Feature engineering is fundamental to the application of machine learning, both difficult and expensive. A generalized data representation is an essential pre-process for feature engineering. The solution provided by the local search approach is a sequence of the movements for all the trains. However, we would like to decompose the solution to movements and each time predict one next movement for current problem state. So, an appropriate representation is crucial to describe the problem state as the input of the further process. It should describe the current location of the trains at tracks, the future tasks to be performed for each train, etc. The problem state is a hidden knowledge and unstructured which makes it a non-trivial task.

There are 2 main entities in the state, namely trains and tracks. An initial idea about the state representation is to store all information in a fixed size matrix (table). However, it will result in a large number of features and a sparse dataset. The high dimensional dataset is likely to cause overfitting in the Decision Tree. Hence, we propose to design a track based feature set to record necessary information based on each track. Then we can fix a general feature set given a shunting yard layout.

The tricky issue is the similarity measurement of the problem states. Because the problem state representation consists of various data types, such as binary, continuous and categorical. We need a classifier that can process different types of features or do pre-processing to normalize the features.

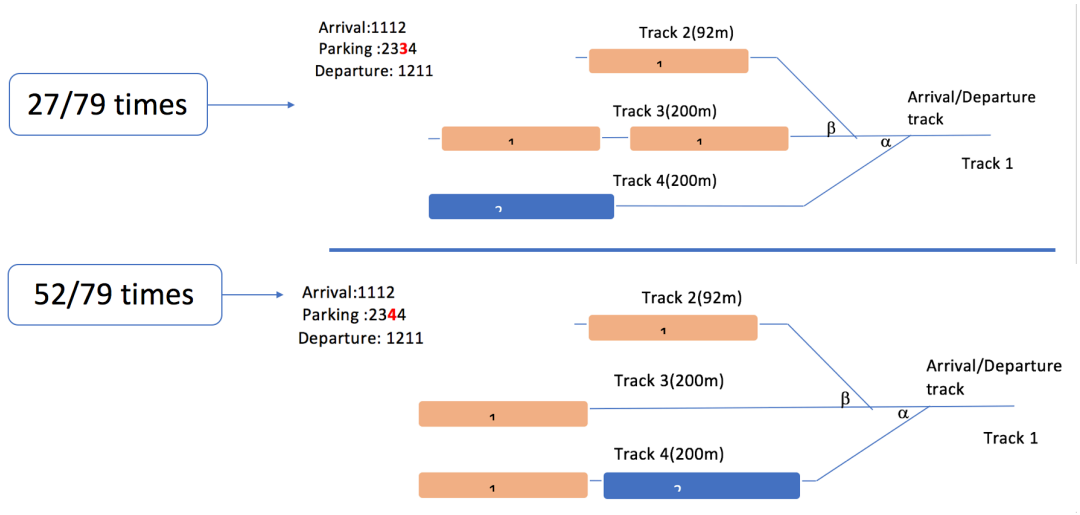


Figure 2.3: The parking assignments for instances in the example set. A yellow block is a train with material_id 1 and a blue one with materail_id 2. The sequence of "Arrival" and "Departure" is the order of train by material id. The sequence of "parking" is the parking assignment indicating the track id in the order of arrivals. The red "3" in "parking: 2334" means the third arrived train is parked at Track 3.

We propose 2 ways of encoding the categorical features; one is to encode by integers, the other is to transform by onehot encoding.

Subquestion: 2. How to model the labels of the features to predict actions for a problem state?

There are several features and one label for each sample in the training data. In this case, the features are problem states and the label for one problem state is the next action to be taken by a train. The amount of the label can be *n*r* at the maximum when there are at most *n* trains and *r* possible actions to choose from. This large amount of target classes makes it difficult to construct an efficient and accurate machine learning model. We need to reduce the prediction space for machine learning models by domain knowledge.

We would like to design an interpretable model which could explain the principle applied to classify the problem states. So, an effective method is needed to compare problem states accurately and reasonably. Inspired by the human's planning process, some features make a more noticeable effect than others in a planning and scheduling task. Consequently, we propose the hierarchical tree model that is both interpretable and efficient.

The main structure of the tree model will be constructed based on the domain knowledge and the feature engineering. There will be decision switches at each node, and most significant features will be considered at nodes closer to the root node. In each leaf node, several similar states are sharing one or more actions. During the planning, the most promising action with the highest frequency will be selected for the state that ends to that leaf node.

Subquestion: 3. How can we combine the proposed approach with Local Search algorithm to generate robust and feasible solutions?

The Local Search algorithm demonstrates the ability to generate feasible solutions for complex problems in few minutes. Out proposed approach is expected to produce robust solutions is a short computation time. One concern to our approach is that it may not be able to solve complex problems. Therefore, we would like to find a method to combine these 2 approaches to utilize their advantages.

3

Literature Review

In this section, a detailed literature review related in this problem scope will be discussed including Train Unit Shunting Problem (TUSP) and Decision Tree.

3.1. Train Unit Shunting Problem

The Train Unit Shunting Problem (TUSP) is first introduced by Freling et al.[2] and it consists of 5 subproblems, namely matching, parking, service scheduling, routing and crew planning. The matching is an assignment of arriving train units to departure positions. A train consists of one or more train units. Splitting or combining of trains may be needed depends on the departure requirements and the matching assignment. For the service scheduling, the cleaning and other short-term maintenance tasks are planned to perform at specific tracks. Routing is a planning of paths of all movements within the service site. The crew planning is the scheduling of all mechanics who are capable of performing specific service tasks. Besides service operation, train units are parking at empty tracks.

One method is to solve the problem by solving each component sequentially. In Freling et al. work [2], they develop a 2-step solution approach for matching and parking subproblems. The matching problem is modeled as a Mixed Integer Programming (MIP) problem and solved by CPLEX solver. A column generation approach is employed to find a feasible parking plan. Then the parking plan is adjusted through dynamic programming so that each train can fit in the assigned track and park on time without being blocked by other trains. Lentink et al. employ a similar approach to Freling et al. [2] but including the routing subproblem and decompose the problem in 4 steps [3]. After the matching assignment in step 1, a graph representation of the shunting yard is generated to estimate the routing cost from and to each track in step 2. These estimations are taken as the input to step 3, the parking subproblem. At last, the actual routing costs are computed given the graph representation and track occupation from previous steps. Later, Lentink extends the problem scope by including one service task, the cleaning [4]. The first 3 subproblems are solved by the same approach in the previous work [3]. Then the cleaning subproblem is modeled as a crew scheduling problem and solved by using CPLEX solver. Van Dommelen develops an approach for matching, parking, routing and service tasks in a decomposed manner [5]. The matching problem is solved firstly. Then the service tasks scheduling is modeled as a flow shop problem and solved by CPLEX solver. Given the parking intervals from the previous step, OPG, a tool developed in NS, computes both the parking locations and the routes. It reduces the computation time for generating acceptable solutions through decomposition, but it also causes the loss of global optimality.

The other method is to invest in integrated approaches that could solve all subproblems simultaneously. The problem complexity increases distinctly in the integrated method. Kroon et al. have investigated an integrated method for the combined matching and parking subproblems [6]. They group the massive amount of crossing constraints in clique constraints in the mathematical formulation. It is practical for a solver to find feasible solutions within a reasonable time with the reduction in constraints. However, the computation time increases significantly in larger problems. Van den Akker

et al. also construct solutions for the integrated matching and parking subproblems [7]. An exact dynamic programming algorithm takes the solutions from a greedy heuristic, and conduct pruning in the dynamic programming network to reduce the computation time. Jekker proposed 2 genetic algorithms (GA) for the combined matching and parking subproblems [8]. Jacobsen and Pisinger construct solutions for the parking and service scheduling through 3 heuristics, Guided Local Search, Guided Fast Local Search and Simulated Annealing [9]. It takes a longer computation time for integrated approaches, from 10 minutes up to hours, due to the increased problem complexity.

The best approach so far in NS is an integrated Local Search approach presented by Van den Broek which solves the 4 subproblems for instances with 23 trains composed of 27 train units in 4 minutes [1]. It is a simulated annealing algorithm that starts with an initial plan. Then, it explores through the search space by evaluating each subproblem and simultaneously constructs the shunting plan. A shortcoming of this approach is the lack of robustness. More specifically, small delays can cause radical changes in the original plan.

3.2. Decision Tree

A Decision Tree classifier is a method applied to construct complex decision-making. It is expressed as a rooted tree that can recursively partition among the instance space which is the set of all possible examples. The root node, without any incoming node, is the start point of the partitioning. All other nodes with only one incoming node are child nodes. Those with no child node are called leaves. A Decision Tree classifier is a model that It represents the general relationship between the input attributes and target attributes [10]. Each internal node is associated with one or more input attributes that is employed to split the instance space into two or more subspaces. Each leaf node is assigned to a class with a most appropriate target value. When classifying the instances with a decision tree, it will navigate the instances from the root node down to a leaf by checking their input attributes in each internal node along the path. The model will output the class assigned to the leaf where the instance is classified.

Typically, the goal is to construct an optimal decision tree with minimum generalization error, which is the misclassification rate over the distribution of the target attributes. Induction of an optimal model from a given training set is NP-hard, and it is only possible in small problems [10]. We can estimate the topology of the tree and the decision rule at each node empirically by using real-world data with the labeled targets, i.e., supervised learning.

An entity, the inducer, performs the induction of the tree structure and the decision rule. Most often the tree induction follows a top-down manner, like in ID3 [11], C4.5 [12], and CART [13]. Generally, the tree is constructed through a growing phase followed by a pruning phase. During tree construction in a top-down manner, a discrete function of the input attributes is utilized to partition the instance space recursively. In each iteration, a most appropriate function is selected according to some split measures. After the selection, each node subdivides the instance space into subspaces considering the outcome of the discrete function, until no split gains sufficient split measures or one stopping criterion is fulfilled. In most cases, there is only a single input attribute as the splitting criterion in the discrete function. Chandra and Verghese [14] mention two commonly used standard splitting measures, Gini index [13] and Gain ratio [12].

The scheduling problem can be modeled as a Rule-Based process by incorporating the scheduling knowledge into IF-THEN rule and form an expert system. This approach also called inductive learning, which can be defined as the process of inferring the description of the class from the description of the individuals belongs to that class [15]. Shaw et al. apply a sequence of the IF-THEN rule to define a class in dispatching rule selection in the manufacturing system to achieve a better overall performance. They first classify the distinct manufacturing patterns, then generate a decision tree with decision nodes related to the system current processing state [15]. The model can dynamically choose the appropriate dispatching rule given a set of system attributes. Then Shiue et al. develop a hybrid model which employs an artificial neural network to identify the significant system attributes and use the decision tree to learn the whole set of the training sets with essential attributes to improve

the knowledge representation [16]. The results demonstrate that the systems attributes identification process delivers better generalization ability.

4

Feature Engineering

We present the process of defining the feature set by comparing the two different feature sets regarding the performance in a toy test in section 4.1. Also, we explain the difference of the integer encoding and the onehot encoding at section 4.2.

4.1. Feature Set

We aim to design a feature set that captures necessary information of current state to make predictions of actions. An appropriate feature set will represent similar states in similar values so that it leads to the same robust solutions. There are 2 main entities to consider in a state, namely trains and tracks. We present 2 feature sets based on trains and tracks respectively. Then we compare their performance on a toy test set.

4.1.1. Train Based Features

Track Occupation & Future Events

- arriving_trainunits: `numerical`. The amount of future arrivals.
- empty_tracks: `numerical`. the amount of empty tracks.
- parking_trainunits: `numerical`. the amount of parking train units.

There are n trains in each problem instances and each train has following 8 features to describe its location, its train material type, and future arriving and departure details.

- train_1_arrival: `continuous`. When the train will arrive? (0–arrived; 151 > 0 : arriving in 151 min)
- train_1_departure: `continuous`. When is the earliest possible time for this train to depart? (it can depart as long as the material_id matches the requirement.)
- train_1_material_id: `categorical`.
- train_1_position: `categorical`. The position of the train unit in this train.(A train may consists of 1-2 train units.)
- train_1_track_id: `categorical`. The track id of the train located.(1–gate track, 2/3/4–parking track)
- train_1_train_id: `categorical`
- train_1_train_position: `categorical`. The location of the train in that track. (2 trains may be parked in one track.)

- train_1_trainunit_id: `categorical`.

Trigger Information

- trigger_type: `categorical`; 1—departure, 2 —arrival.
- trigger_material: `categorical`, the value is material_id.
- trigger_train: `categorical`, the value is train_id.
- state_id: `numerical`, the value is the current state index.

4.1.2. Track Based Features

This feature set consists of track occupation, future events, and trigger information. The track occupation part contains information about whether a track is occupied and by what material type of train. The following list gives an overview of the feature set we designed for the shunting yard used in experiments. There are $r = 3$ parking tracks, and each of them can park 1 to 2 train units. Suppose there are n arrivals and n departures in each problem instances.

Track Occupation

- track_2_position_0: `categorical`. The value is train material_id.
- ...
- track_4_position_1
- arriving_trainunits: `numerical`. The amount of future arrivals.
- empty_tracks: `numerical`. The amount of empty tracks.
- parking_trainunits: `numerical`. The amount of parking train units.

Future Events

- next_arrival_1: `categorical`. The value is train material_id.
- ...
- next_arrival_n
- next_departure_1: `categorical`. The value is train material_id.
- ...
- next_departure_n

Trigger Information

- trigger_type: `categorical`; 1—departure, 2 —arrival.
- trigger_material: `categorical`. The value is material_id.
- state_id: `numerical`. The value is the current state index.

4.1.3. Comparison of Feature Sets

We test the algorithms with 2 different feature sets on a toy problem and compare the performance according to the number of instances solved. There 200 instances in this test set, and there are 4 trains in each instance. The shunting yard contains 3 dead-end parking tracks and one gate track. From the results in Table 4.1, the algorithm with tracked based features solves much more instances than that with train based features. It shows the track based features is more beneficial to this scheduling problem. Hence we decide to use the track based feature set as the feature set for this scheduling problem.

algorithm	DTS with train based features	DTS with train based features
solved instances	133	200

Table 4.1: Comparison the performance of different feature sets in terms of the number of instances solved in the toy test.

4.2. Onehot Encoding for Categorical Features

There are several categorical features in the proposed feature set such as "next_arrival_1". They are encoded as integers in our dataset. Typically, a machine learning model assumes that there exists a natural ordering in the numerical values, such as "3" is larger than "1". It may lead to misclassification with this assumption when there is no such relation in the features. Therefore, we propose to transform the categorical features by onehot encoding. Here is an example explaining the difference between categorical features and onehot encoded features.

Suppose there is 4 possible values for the feature "next_arrival_1", and they have been encoded as $(1, 2, 3, 4)$ as in Table 4.2. By onehot encoding, there will be 4 new features, and the values are binary as in Table 4.3. If the value in the feature "next_arrival_1" equals to "1", then in the new columns, the value of "next_arrival_1 == 1" is 1. There is only one column has value equals 1 in the onehot encoded features. In this way, we avoid the assumption of natural ordering in integer encoded categorical features. It may lead to more accurate prediction and result in better overall performance.

Table 4.2: Categorical feature in integers. The value to "next_arrival_1" is the material type id of that arrival train.

index	next_arrival_1
1	1
2	2
3	3
4	4

Table 4.3: Onehot encoded features for "next_arrival_1".

index	next_arrival_1 == 1	next_arrival_1 = 2	next_arrival_1 == 3	next_arrival_1 == 4
1	**1**	0	0	0
2	0	**1**	0	0
3	0	0	**1**	0
4	0	0	0	**1**

5

Additional Experiments

In this chapter, we explain the detailed results and observations in 2 scenarios. We validate our proposed algorithm in a toy scenario, scenario 1 in section 5.1. In section 5.2 we analyze the construction of Decision Trees for Scenario 2 (the same scenario in the scientific paper). For each scenario, we design 2 subsets of 200 instances of medium and difficult according to the difficulty level of solving the problems. For a problem instance of medium level, the number of trains is slightly larger than the number of tracks. In a few tracks, there are 2 or more trains parking that results in certain flexibility in solutions. In difficult problem instances, there are more trains and the usage of track length is rather high. For each instance, there are only a few solutions with limited flexibility.

5.1. Scenario 1

In this scenario, we design a shunting yard with three First In Last Out tracks and one gate track. There are two types of train material involved, namely ICM-3 and ICM-4, of which details are in table 5.1. There are two subsets of the problem instances, one with 4 train units and the other with 5. For this layout, there are five parking positions, one at track 1 and two at track 3 and 4 respectively. For the future events, we provide the next k = 5 arrivals and departures individually. The target set is the set of all parking tracks (2,3,4). We generate 800 problem instances for each problem size as training data.

For instances with 4 train units, the train units set composes of 3 trains with material ICM-3 and one with material ICM-4. There are 16 possible combinations of the arriving and departure sequence by ignoring the exact time. The usage of total parking track length is 71.7%. We consider it as a medium instance according to the difficulty level. For instances with 5 train units, there are 3 trains with material ICM-3 and two with material ICM-4. It can make 100 different combinations of the arrival and departure sequence. All trains occupy 91.9% of the parking track length in total. Problems are relatively tricky in this set with 5 train units.

Criteria:
We compare the performance regarding the number of instances solved (NF), the number of unique solutions (NU), and the ratio of unique solutions to feasible solutions (RUF). The algorithm is considered the best if it solves more problem instances with fewer unique solutions.

Baseline Algorithms:
We conduct experiments among the proposed DTS, DTS_1 (onehot encoded features), a heuristic (designed based on the algorithm in [17]) and the Local Search algorithm [1]. The heuristic algorithm is expected to be rather robust that solves problems with fewer unique solutions. The Local Search will solve most problems but with more unique solutions.

5.1.1. Results

In this scenario with 4 trainunits, DTS and DTS 1 solve the same amount of problem instances with the same amount of unique solutions from table 5.2. They solve all the problem instances as LS but

Table 5.1: Train unit information in Scenario 1. The arriving ratio shows the appearance probability of a train with that material type in one instance.

material type	train length (m)	arrival ratio
ICM-3	81	0.64
ICM-4	107	0.36

Table 5.2: Performance comparison in Scenario 1. The ratio is between the number of unique solutions and the number of feasible solutions.

Alg		DTS	DTS_1	Heuristic	LS
	solved	200	200	163	200
size_4	unique	6	6	4	30
	ratio (%)	3	3	2.4	15
	solved	157	162	36	162
size_5	unique	15	14	4	30
	ratio (%)	9.6	8.6	11.1	18.5

outperform LS with fewer unique solutions. The heuristic produces the fewest unique solutions and solves the fewest problem instances.

When the problem size increases to 5 train units, DTS 1 solves the same amount of problem instances as LS but with a half amount of unique solutions. DTS underperforms slightly than DTS 1 with 5 fewer solved instances and 1 more unique solutions. The heuristic again solves the smallest amount of problem instances with the fewest unique solutions.

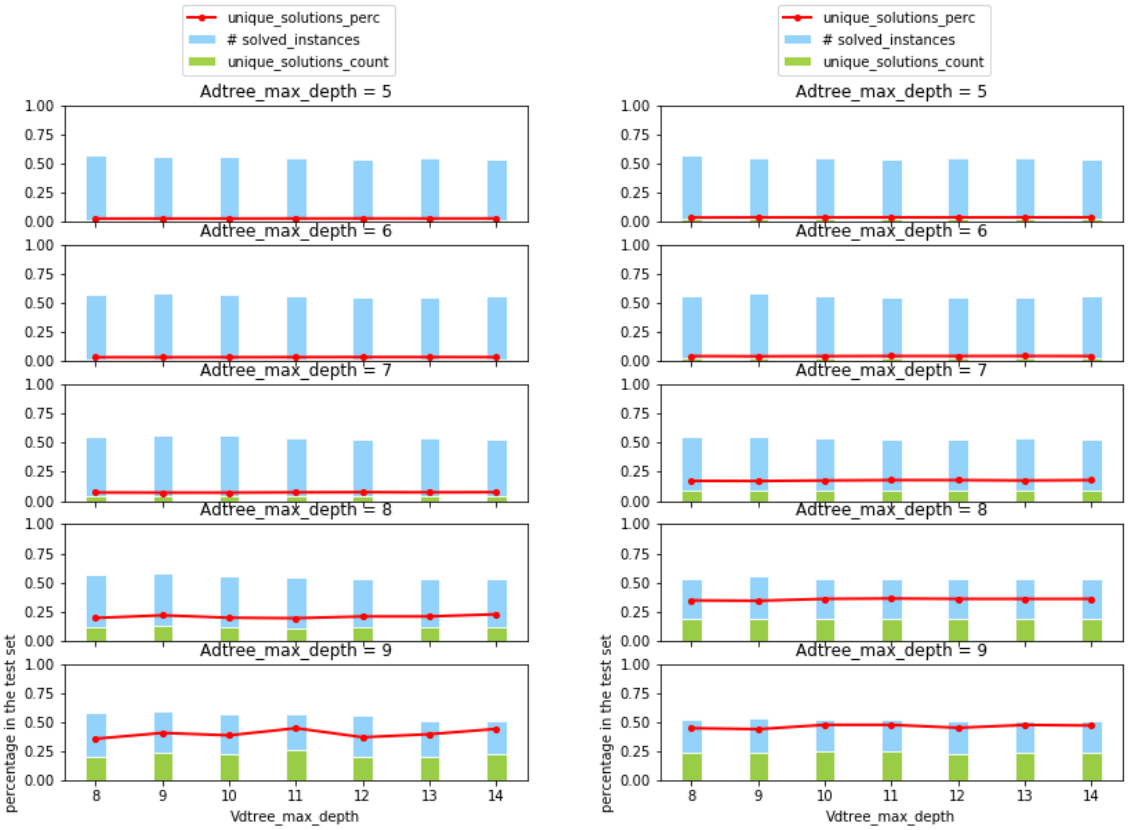
5.2. Scenario 2

In this scenario, we test our approach in a shunting yard with 9 First In Last Out tracks with trains distributed in 6 different material types. There are 2 subsets of instances, each of them contains 10 and 12 trains respectively. We generate the 2000 problem instances for each subset as training data. We also set 200 instances for validation among different Decision Tree settings because of the following reasons. First, the problem size and complexity increasing leads to the extension of the training set volume, the feature set, and the target class set. It may result in DT overfitting and construct a complex tree with less generalization ability. Second, when the tree size goes enormous, it loses the interpretability since it is too complicated to read the rules. For these reasons, we set a stopping criterion, the max tree depth, to constrain the tree structure. In the remaining section, we study the impact of different DT models in the performance of solving problems in the validation set. We experiment the impact of max tree depth on the algorithm's performance regarding the number of instances solved (NF), the number of unique solutions (NU), and the ratio of unique solutions to feasible solutions (RUF).

5.2.1. Scenario 2 with 10 trainunits

DTS performance

In Figure 5.1 at left, in each row with a fixed arrival DT max depth d_A, the blue bars fluctuate slightly in the range of 51% to 59%. The number of instances solved (NF) reaches the peak when the departure DT max depth $d_V = 8$ or 9 (the highest blue bar). The number of unique solutions (NU) does not diversify obviously, so the lowest RUF locates at d_V =8 or 9. For a fixed d_V (e.g., $d_V = 8$), NF fluctuates for different d_A in a range of 55% to 57%, but NU arises dramatically as d_A increases(see the green bar rises in a column). The ratio of unique solutions to feasible solutions RUF shows a huge jump from the top row to the bottom. To sum up, we select $(d_A = 5, d_V = 8)$ to train the DT models in DTS for the instances subset with 10 train units. The model with that setting solves 113 instances with 3 unique solutions in the validation set of 200 instances.



(a) categorical features **(b)** onehot encoded features

Figure 5.1: Comparing the performance of DTS with different DT settings in Scenario 2 with 10 train units. For trees with a larger max depth, the model can solve more problems as the blue bar raises from top to down also left to right. However, the number of unique solutions also increases with trees with a larger max depth as the green bar grows. It indicates the decreasing of consistency in solutions.

DTS_1 performance

In Figure 5.1 at right, in each row with a fixed arrival DT max depth d_A, the blue bars fluctuate slightly in the range of 51% to 58%. The number of instances solved NF reaches the peak when the departure DT max depth $d_V = 8$ or 9 (the highest blue bar). NU does not differ obviously, so the lowest RUF locates at d_V =8 or 9. For a fixed d_V (e.g., $d_V = 8$), NF fluctuates for different d_A in a range of 53% to 57%, but NU arises dramatically as d_A increases(see the green bar rises in a column). The ratio of unique solutions to feasible solutions RUF shows a huge jump from the top row to the bottom. To sum up, we select $(d_A = 5, d_V = 8)$ to train the DT models in DTS_1 for the instances subset with 10 train units. The model with that setting solves 113 instances with 4 unique solutions in the validation set of 200 instances.

5.2.2. Scenario 2 with 12 trainunits

DTS performance

In Figure 5.4 at left, in each row with a fixed arrival DT max depth d_A, the blue bars fluctuate slightly and the difference is in between 3% to 10%. The number of instances solved (NF) reaches the peak when the departure DT max depth $d_V = 8$ or 9 (the highest blue bar). The number of unique solutions (NU) does not vary obviously, so the lowest RUF locates at d_V =8 or 9. For a fixed d_V (e.g., $d_V = 8$), NF fluctuates for different d_A in a range of 35% to 44%, but NU arises significantly as d_A increases(see the green bar rises in a column). The ratio of unique solutions to feasible solutions RUF shows a sharp increase from the top row to the bottom. To sum up, we select $(d_A = 5, d_V = 8)$ to train

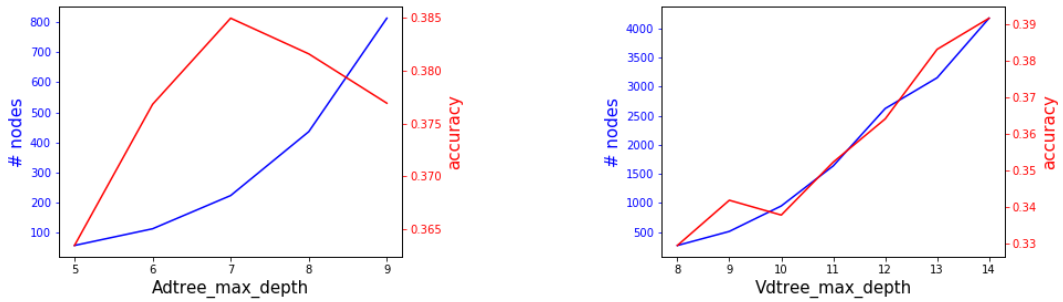


Figure 5.2: Comparison of Decision Trees for arrival events (left) and departure events (right) with different settings in Scenario 2 with 10 train units. The total number of nodes is in blue, and the prediction accuracy is in red.

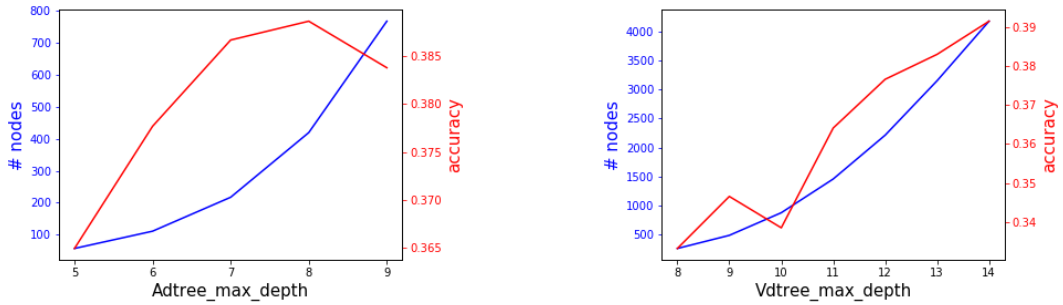


Figure 5.3: Comparison of Decision Trees for arrival events (left) and departure events (right) with different settings in Scenario 2 with 10 train units (onehot encoded features). The total number of nodes is in blue, and the prediction accuracy is in red.

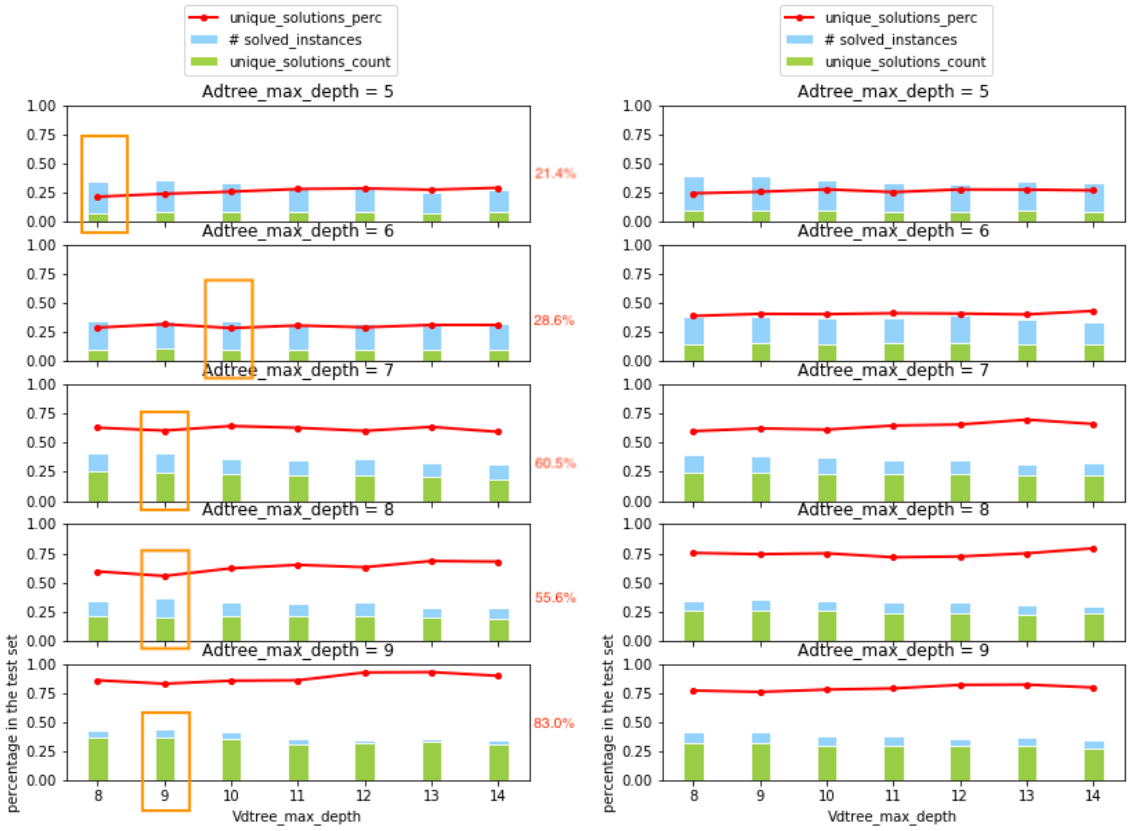
the DT models in DTS for the instances subset with 12 train units. The model with that setting solves 70 instances with 15 unique solutions in the validation set of 200 instances.

DTS_1 performance

In Figure 5.4 at right, in each row with a fixed arrival DT max depth d_A, the blue bars fluctuate slightly and the difference is in between 5% and 8%. The number of instances solved NF reaches the peak when the departure DT max depth $d_V = 8$ or 9 (the highest blue bar). NU does not vary obviously, so the lowest RUF locates at d_V =8 or 9. For a fixed d_V (e.g., $d_V = 8$), NF fluctuates for different d_A in a range of 35% to 42%, but NU arises dramatically as d_A increases(see the green bar rises in a column). The ratio of unique solutions to feasible solutions RUF shows a huge jump from the top row to the bottom. To sum up, we again select $(d_A = 5, d_V = 8)$ to train the DT models in DTS_1 for the instances subset with 12 train units. The model with that setting solves 78 instances with 19 unique solutions in the validation set of 200 instances.

5.2.3. Summary

We conduct experiments in 2 subsets of problem instances to investigate the impact of different tree depths on the overall performance. Generally, a larger arrival DT model leads to more unique solutions and a smaller departure DT model results in solving more problems. Although the prediction accuracy of a smaller DT model in lower than that of a larger one. At last, we deliver a combination of $(d_A = 5, d_V = 8)$ to all subsets and both DTS and DTS_1 for the best performance of solving more problems with fewer unique solutions.



(a) origin features **(b)** onehot encoded features

Figure 5.4: Comparing the performance of DTS with different DT settings in Scenario 2 with 12 train units. For trees with a larger max depth, the model can solve more problems as the blue bar raises. However, the number of unique solutions also increases with trees with a larger max depth as the green bar grows. It indicates the decreasing of consistency in solutions.

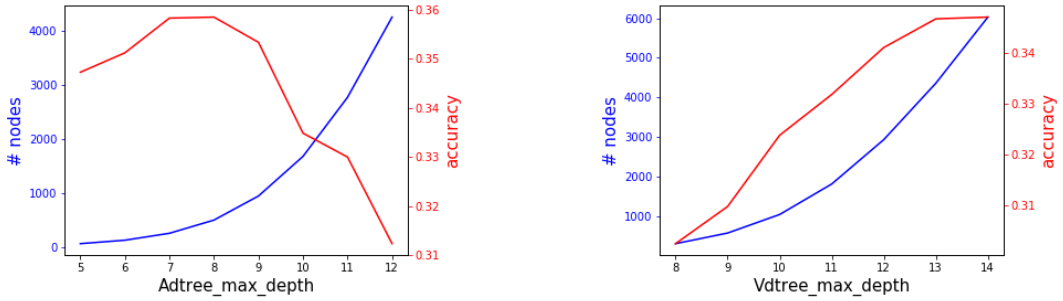


Figure 5.5: Comparison of Decision Trees for arrival events (left) and departure events (right) with different settings in Scenario 2 with 12 train units. The total number of nodes is in blue, and the prediction accuracy is in red.

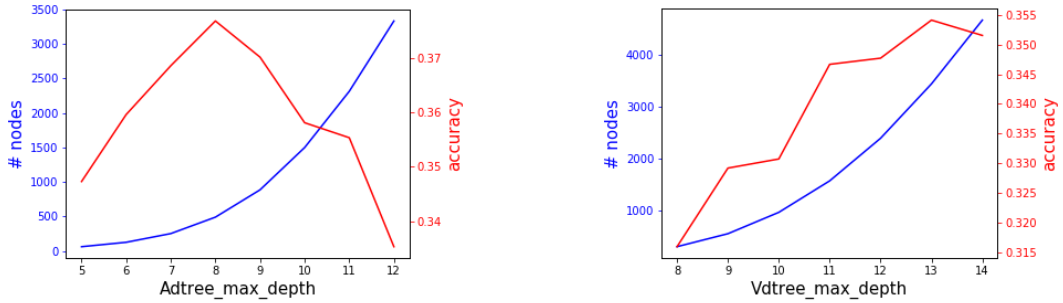


Figure 5.6: Comparison of Decision Trees for arrival events (left) and departure events (right) with different settings in Scenario 2 with 12 trainunits(onehot encoded features). The total number of nodes is in blue and the prediction accuracy is in red.

6

Conclusions and Discussions

6.1. Conclusions

Many researches have succeeded in finding feasible solutions to TUSP. However, few of them focus on the robustness of solutions. In reality, it is preferred to obtain robust solutions to deal with uncertainty such as arrival delays. To this end, we developed a sequential approach to solve TUSP using machine learning techniques, more specifically, the Decision Tree Classifier. We conduct extensive experiments to compare the performance of the proposed approach, a heuristic algorithm, and the Local Search algorithm. Our proposed approach outperforms those benchmark algorithms according to the robustness although it solves fewer problems than others in the scenario with more trains. DTS and DTS_1 demonstrate similar performance in both test sets indicating that the feature encoding method has no apparent effect on the final results in this problem.

Although we only conducted experiments on one shunting yard, this framework can be transformed and applied to other similar shunting yards with some adjustments. As long as we have historical data about problem instances and solutions, we can construct appropriate Decision Trees to make predictions and build solutions. This sequential framework can also be generalized to other scheduling problems that require robustness in solutions, such as metro or tram scheduling. The critical issue is to design an appropriate feature set.

6.2. Research Questions

We discuss and answer the research questions through experiment results in this section.

Can Decision Trees improve the robustness in solutions of the train unit shunting problem?

We have demonstrated our approach by transforming historical solutions into data sets for learning decision trees, which subsequently are used to predict the actions of arrivals and departures for new train shunting problems. We ran an extensive set of experiments with different problem sizes. Although solving train shunting problem is a hard problem, our proposed methods obtained feasible solutions for 50% problems with very few unique solutions, significantly outperforming other approaches from the literature concerning robustness.

Subquestion: 1. How to represent a problem state for train unit shunting problem?

Those features to the machine learning model are intended to capture necessary information for predicting valid actions and making schedules. We designed a track based feature set containing information about track occupation, future events, and trigger information. There are different types of features, e.g., binary and categorical. We implement 2 versions of the feature set; one encodes the categorical features as integers, the other transforms categorical data by onehot encoding. From

the overall performance, there is no apparent difference caused by different encoding methods. One advantage of using onehot encoding is that it brings more interpretability by avoiding the assumption of natural ordering in integer encoded categorical features.

Subquestion: 2. How to model the labels of the features to predict actions for a problem state?

We designed a hierarchical structure to reduce the prediction space. Firstly, we use separate DT models to predict actions for arrivals and departures. Secondly, for an action, we predict only one track id by fixing the target train. In this way, we determined the target class as the parking track set in the shunting yard.

Subquestion: 3. How can we combine the Local Search algorithm with proposed approach to generate feasible and robust solutions to TUSP?

The proposed approach is promising in generating robust solutions in a few seconds but fails to produce feasible solutions to complex problems with more trains. The Local Search is capable of solving complex problems. Therefore, we propose an idea of using 2 approaches in combination to achieve better results. For an instance, we first run the proposed algorithm to find a robust solution. If no solution found, we use the Local Search algorithm instead. From the experiment results, we notice this combination demonstrates the strong ability to solve problems with robust solutions.

6.3. Limitation and Future Work

Solving More Problem Instances:

The proposed DTS and DTS_1 solve relatively fewer instances than other methods. We list the possible reasons and discuss potential solutions. One reason is that there exists randomness in the solutions from LS. The labels attached to the samples are not guaranteed to be the ground truth in the training data. However, DT infers the decision rules considering those labels which makes it difficult to induce an accurate DT model. Our suggestion is to collect less random solutions as training data to the DT model. For example, assemble the solutions generated by DTS to reduce the noise in the training set.

The second reason is the errors accumulation during the sequential scheduling. The algorithm fails when it can not find a valid action at a state, but it may result from an inappropriate action earlier. This influence can be reduced by allowing some recovery mechanisms. Currently, the algorithm has several chances to choose another action if it fails at the first attempt at that state. One option is to extend the recovery chances to one or two states back. If it can not find any valid action in the current state, then we allow it to take a different action in the previous state. The other option is to allow the temporary relocation in the shunting yard. Mostly the algorithm fails when parking the last arriving train or departing blocked by trains parking in front. If it can temporarily relocate trains when necessary, then we could find feasible solutions for more problem instances.

The third reason could be that we make a tradeoff between solving more instances and generate fewer unique solutions during DT models construction. A small DT model loses some details to plan less common cases, but it leads to robust solutions. One way is to combine the proposed approach with the Local Search algorithm to utilize their advantages. From the experiments, we verified this combination could solve more problems with robust solutions.

Extend the Problem Scope:

The current structure does not take the service scheduling into account. So we could not yet apply it directly in practice. One possible application can be generating initial solutions to the Local Search algorithm. Our proposed approach can generate robust solutions for TUSP without the service scheduling. We expect a robust initial solution can lead to a robust final solution.

Bibliography

[1] R. W. van den Broek, *Train Shunting and Service Scheduling: an integrated local search approach*, Master's thesis (2016).

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

[3] 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 , 415 (2006).

[4] R. Lentink, *Algorithmic decision support for shunt planning*, 73 (2006).

[5] M. T. van Dommelen, *Scheduling train service activities at service sites to determine the capacity*, Master's thesis (2015).

[6] L. G. Kroon, R. M. Lentink, and A. Schrijver, *Shunting of passenger train units: an integrated approach,* Transportation Science **42**, 436 (2008).

[7] 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,* (2008).

[8] D. Jekkers, I. U. Kaymak, and L. G. Kroon, *Train shunt planning using genetic algorithms*, Master's thesis (2009).

[9] P. M. Jacobsen and D. Pisinger, *Train shunting at a workshop area,* Flexible services and manufacturing journal **23**, 156 (2011).

[10] L. Rokach and O. Maimon, *Top-down induction of decision trees classifiers-a survey,* IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **35**, 476 (2005).

[11] J. R. Quinlan, *Induction of decision trees,* Machine learning **1**, 81 (1986).

[12] J. R. Quinlan, *C4. 5: programs for machine learning* (Elsevier, 2014).

[13] L. Breiman, *Classification and regression trees* (Routledge, 2017).

[14] B. Chandra and P. P. Varghese, *Moving towards efficient decision tree construction,* Information Sciences **179**, 1059 (2009).

[15] M. J. Shaw, S. Park, and N. Raman, *Intelligent scheduling with machine learning capabilities: the induction of scheduling knowledge,* IIE transactions **24**, 156 (1992).

[16] Y.-R. Shiue and C.-T. Su, *An enhanced knowledge representation for decision-tree based learning adaptive scheduling,* International Journal of Computer Integrated Manufacturing **16**, 48 (2003).

[17] E. Beerthuizen, *Optimizing train parking and shunting at NS service yards*, Master's thesis (2018).