Transportation Research Part C 137 (2022) 103566

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

Estimating the robustness of public transport schedules using machine learning☆

Matthias Müller-Hannemann [a], Ralf Rückert [a], Alexander Schiewe [b,*], Anita Schöbel [b,c]

[a] Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, Halle (Saale), Germany
[b] Fraunhofer Institute for Industrial Mathematics ITWM, Kaiserslautern, Germany
[c] University of Kaiserslautern, Kaiserslautern, Germany

ARTICLE INFO

Keywords:
Public transportation
Scheduling
Timetabling
Machine learning
Robustness
Optimization

ABSTRACT

The planning of attractive and cost efficient public transport schedules, i.e., timetables and corresponding vehicle schedules is a highly complex optimization process involving many steps. Integrating robustness from a passenger's point of view makes the task even more challenging. With numerous different definitions of robustness in the literature, a standard way to evaluate the robustness of a public transport system is to simulate its performance under a large number of possible scenarios. Unfortunately, this is computationally very expensive.

In this paper, we therefore explore a new way of such a scenario-based robustness approximation by using regression models from machine learning. Training of these models is based on carefully selected key features of public transport systems and passenger demand. The trained model is then able to predict the robustness of untrained instances with high accuracy using only its key features, allowing for a robustness oracle for transport planners that approximates the robustness in constant time. Such an oracle can be used as black box to increase the robustness of public transport schedules. We provide a first proof of concept for the special case of robust timetabling, using a local search framework. In computational experiments with different benchmark instances we demonstrate an excellent quality of our predictions.

1. Introduction

Public transport planning is a challenging process. The resulting public transport schedules, i.e. timetables and vehicle schedules, should not only be cost efficient and offer a high service quality to passengers, but should also be robust (Lusby et al., 2018), that is, they should be able to compensate for commonly occurring delays. An overview of current challenges in public transport planning can be found in Borndörfer et al. (2018b). In this paper, we focus on the robustness against typical delays (but do not consider resilience and vulnerability of public transport schedules with respect to long-lasting major disruptions).

There are numerous different robustness concepts (for an overview see Goerigk and Schöbel, 2016) from which many have been motivated by timetabling. Among them is recovery robustness (Liebchen et al., 2009; D'Angelo et al., 2009), light robustness (Fischetti and Monaci, 2009), adjustable robustness (Polinder et al., 2019), recovery-to-optimality (Goerigk and Schöbel, 2014; Lu et al., 2017), a bicriteria approach (Schöbel and Kratz, 2009), or an approach focusing on critical points (Andersson et al.,

☆ This work has been partially supported by DFG, Germany under grants SCHO 1140/8-2 and MU 1482/7-2.
* Corresponding author.
E-mail addresses: muellerh@informatik.uni-halle.de (M. Müller-Hannemann), ralf.rueckert@informatik.uni-halle.de (R. Rückert), alexander.schiewe@itwm.fraunhofer.de (A. Schiewe), schoebel@mathematik.uni-kl.de (A. Schöbel).

https://doi.org/10.1016/j.trc.2022.103566
Received 18 May 2021; Received in revised form 17 December 2021; Accepted 13 January 2022
Available online 9 February 2022

2013). An experimental comparison of different concepts is provided in Goerigk and Schöbel (2010). A recent overview is given in Lusby et al. (2018). Each of these robustness concepts uses different ways for evaluating the outcome. The importance of taking the passenger perspective into account has been stressed by Parbo et al. (2016). Therefore, a reliable method for the evaluation of the robustness of a timetable should be based on simulating scenarios and their outcome for the passengers. Measuring the effect of delays for passengers has been provided, for example, in De-Los-Santos et al. (2012), Cats (2016), and Lu (2018). Friedrich et al. (2017, 2018) simulate delays and measure the occurring changes in arrival time for passengers.

A full scenario-based evaluation of robustness, however, is computationally expensive and cannot be done as part of a sub-process in timetable creation, especially because the set of possible delays is very large. In practice, this leads to a timetable that is often either too volatile for occurring delays, is slow in terms of expected travel time, or too cost-inefficient.

In this paper, we consider different kinds of public transport including trains, trams, buses, and the like.

Goals and contribution. The long-term goal of this work is to improve the planning of robust public transport systems using a new approach. A public transport system consists of an infrastructure network, a line plan, a timetable and a vehicle schedule. Here, we focus on robust timetables and their corresponding vehicle schedules. We develop an oracle that produces a prediction for otherwise computationally expensive robustness calculations using simulations. The oracle evaluates the robustness of public transport schedules. Using key features of public transport systems, a machine learning-based estimator should produce robustness values with high accuracy. Selecting these key features and evaluating which are of significance for the oracle will be of importance. We focus on the following research questions:

- Can we predict the robustness of a given public transport system subject to delays by means of machine learning?
- What are suitable features for such predictors and which accuracy can be achieved? Which sample size is needed?
- What are the most important features of a public transport system, contributing to the ability to predict its schedules' robustness?

The machine learning task amounts to solve regression problems. We investigate a variety of approaches, including artificial neural networks, decision tree regression, support vector regression, and several boosting regression methods. Our experimental results show that most of the examined techniques lead to reasonably good approximations, but support vector regression performs best. We demonstrate our results using the robustness evaluation of Friedrich et al. (2018), but our learning approach can also be applied to other evaluation routines. We only require a method which assigns some robustness value to any given public transport schedule. Based on experiments with real-world and artificial benchmark instances we show that an excellent estimation of robustness measures for several robustness tests can be achieved.

We also present an application of such a robustness predictor. Namely, we propose a local search (heuristic) framework for increasing the robustness of a public transport schedule by adapting its timetable that utilizes the oracle as a black box. For an appropriate definition of a neighborhood between feasible public transport schedules, i.e. solutions from an optimization point of view, the basic procedure is simple: In each step, we use the oracle to evaluate all neighbors of the current solution with respect to their robustness and then proceed with the most promising of them. The corresponding research questions are:

- Can local search based on a robustness oracle lead to better timetables?
- How well do estimated and real robustness agree on instances generated by local search?

As we will see, our first attempts to apply a simple local search based iterative improvement scheme are already quite encouraging, leading to the generation of timetables with a reasonable trade-off between perceived travel time for passengers and robustness.

Related work. Public transport planning is an area of ongoing research. It consists of several stages, traditionally solved sequentially with different objective functions. In this work, we consider public transport schedules, consisting of a timetable and a vehicle schedule, with a fixed line plan.

Given stops or stations and their direct connections, *line planning* describes the process of finding bus or train lines that should be served regularly. Besides the goal to transport all passengers, there are many different objective functions to consider, e.g. operational costs, see e.g. Claessens et al. (1998), the number of direct travelers, see e.g. Bussieck et al. (1997), or the travel time of the passengers, see e.g. Schöbel and Scholl (2006). For an overview of different line planning models, see Schöbel (2012).

Afterwards, *timetabling* aims at assigning departure and arrival times to all departures and arrivals of lines at their stations. This can be done periodically, see, e.g. Serafini and Ukovich (Serafini and Ukovich, 1989), or aperiodically, see, e.g. Cacchiani et al. (2010). Especially, finding periodic timetables is a hard problem. There are different approaches for finding suitable solutions for the passengers, see, e.g. Peeters and Kroon (2001), Grafe and Schöbel (2021), or Lindner et al. (2021). For an overview of periodic timetabling in railway planning, see Lusby et al. (2011).

The last part of public transport schedules that we consider here are *vehicle schedules*. Vehicle scheduling describes the problem of finding suitable routes for the available vehicles, serving all lines within a given time interval. Here, we consider aperiodic vehicle scheduling, for an analysis of the differences between periodic and aperiodic vehicle scheduling, see Borndörfer et al. (2018a). For an overview of different models in vehicle scheduling we refer to Bunte and Kliewer (2009).

There are many papers providing approaches for finding *robust* timetables for the different robustness concepts mentioned above. Since the resulting problem is computationally intractable, most of the approaches focus on heuristics, work on simplified models without considering adapted passenger routes or vehicle capacities, or treat special cases such as tree networks. We again refer to the survey of Lusby et al. (2018), recent approaches include Polinder et al. (2020) or Pätzold (2021).

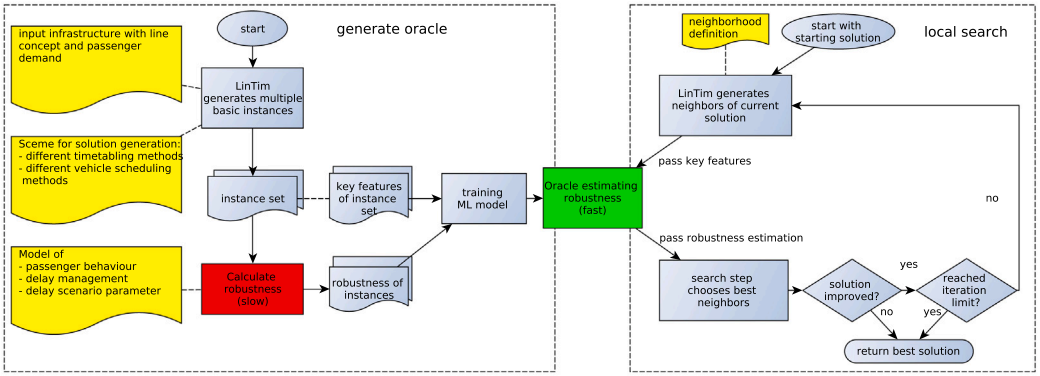


Fig. 1. Workflow for creating the oracle (left box, see Section 3) and using the oracle within local search (right box, see Section 4). Yellow fields denote input or choices of models and methods which are specific for each application. In detail:
Left Box: Creating the oracle for estimating robustness of a public transport schedule by training a machine learning model. This step is only done once for each dataset: For a given infrastructure, a line concept and a passenger demand, we generate many different timetables and for each of them many vehicle schedules and corresponding passenger routes for training, validation, and testing. Afterwards the oracle can be used for black-box optimization of public transport schedules.
Right Box: Local search as an exemplary application of the oracle. With each iteration through the optimization loop, we try to increase the robustness of the instance. Once the local neighborhood does not contain a more robust solution, the optimization terminates. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

We want to point out that in comparison to these and other approaches in literature our main goal in this paper is not the isolated optimization of one robust public transport schedule itself. It is to allow for an extremely fast approximation of the robustness of a whole instance which can later be used as a black box for optimization algorithms. This is similar to Wüst et al. (2019), where the authors use a black box based on max-plus algebra to determine the robustness of a timetable and propose an iterative solution process in order to improve the robustness of the timetable. Here, we develop such an oracle based on machine learning, not only considering the timetable but also the corresponding vehicle schedule. The latter has an important impact on robustness, for example, if delays of one trip carry over to the subsequent trip served by the same vehicle.

Most robustness evaluations are based on the travel times of the passengers. If delays occur, these travel times depend on the delay management strategies, e.g., if a train waits for transferring passengers or if it departs on time. This is the topic of *delay management*, for an overview we refer to Dollevoet et al. (2018) and König (2020). In this paper, we assume the delay management strategy as given and fixed. Major changes in the delay management strategy have an impact on the passengers' delays and hence on the robustness evaluation. Since our approach can be applied for any delay management strategy, such a change can easily be adopted by a new training of the used predictor.

The field of machine and deep learning currently advances very rapidly. This also impacts the field of public transport, but most papers here do not address the optimization of public transport schedules. Recent domain review articles referencing the uses of deep learning in public transport include traffic flow forecasting, traffic signal control, automatic vehicle detection, traffic incident processing, travel demand prediction, autonomous driving, and driver behaviors, see Nguyen et al. (2018), Wang et al. (2019), and Varghese et al. (2020). There are also first algorithmic approaches, see, e.g., Matos et al. (2021) for using reinforcement learning in periodic timetable optimization without robustness, or Bauer and Schöbel (2014) who used machine learning in delay management, learning up to which delay a connection should be maintained.

Machine learning approaches are also used in the area of delay prediction. Oneto et al. (2018) build a data-driven train delay prediction system in an online setting based on learning algorithms for extreme learning machines. Yap and Cats (2020) consider the problem of predicting disruptions and their impact at specific stations, while Cats and Jenelius (2018) predict the impact of a delay after its occurrence. Another variant is the prediction where in the network delays will occur, see, e.g., Cats et al. (2016) or Yap et al. (2018). In contrast to these papers we do not aim to predict delays, but quantify the overall robustness of a public transport schedule where a large set of delay scenarios is given.

Overview. The paper is structured as follows: First, in Section 2, we give an introduction to public transport scheduling and robustness. Afterwards, in Section 3, we introduce the key features we use and develop a corresponding machine learning model. Additionally, we introduce a proof-of-concept local search algorithm to show the practical relevance of the presented model in Section 4. In Section 5 we evaluate the machine learning model on different datasets, investigate its behavior and the parameter choices of Section 3 as well as the performance of the local search algorithm. In the end, we give a conclusion of our work in Section 6.

Fig. 1 shows a graphical overview of the different workflows presented in this work.

2. Public transport schedules and their robustness

2.1. Public transport scheduling

In order to explain our approach we first introduce the public transport data the model operates on. This is depicted in Fig. 1. We assume the following data to be given and fixed:

- the infrastructure, consisting of stops or stations and direct connections (in the following, we will always use the term stations),
- the line concept (a set of lines with corresponding frequencies),
- static passenger demand in form of an origin–destination matrix.

We call the combination of infrastructure, line concept and passengers' demand a *dataset*. Within our machine learning approach, for a given dataset, we generate the following schedules:

- many timetables (by using different timetabling methods),
- many vehicle schedules for each timetable (by using different vehicle scheduling methods), and, finally
- one set of corresponding passengers' routes (i.e., which lines to use at which times) for each timetable (by using shortest path algorithms optimizing the utilities of the passengers).

The final result which then consists of the infrastructure, the line concept, the passengers' demand together with a timetable, a vehicle schedule, and corresponding passenger routes is called an *instance*.

In the following we give more details on the underlying network structure, how we generate delays and simulate their effects on passengers and explain our models for passenger behavior.

Event-activity networks. We use an event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ (Serafini and Ukovich, 1989; Müller-Hannemann and Rückert, 2017), containing nodes (*events* $e \in \mathcal{E}$) for every arrival or departure of a vehicle at a station and directed edges (*activities* $a \in \mathcal{A}$) between them. There are different types of activities representing the different relations between events, namely *drive* activities for an interruption-free drive between two stations, *wait* activities for the dwelling of a vehicle at a station, *transfer* activities for passengers to change vehicles, and *turnaround* activities for vehicles to reach their next trip. A *trip* is a path of drive and wait activities in the event-activity-network that needs to be operated by a single vehicle.

For each activity there is a feasible time interval given, which is dependent on the infrastructure, i.e., there are lower and upper time bounds $L_a \leq U_a$ for all $a \in \mathcal{A}$. A *periodic timetable* with period T, e.g., $T = 60$ minutes, now assigns a time π_e to every event $e \in \mathcal{E}$ such that the corresponding activity bounds are satisfied, i.e.,

$$(\pi_j - \pi_i - L_a) \mod T \in [0, U_a - L_a] \quad \forall a = (i,j) \in \mathcal{A}.$$

We call $(\pi_j - \pi_i - L_a) \mod T$ the slack of activity $a = (i,j)$.

Given an event-activity network with a periodic timetable, we can roll out the periodic plan to an aperiodic event-activity network, covering not just one planning period but e.g. the whole day, by repeating the periodic event-activity network multiple times.

Based on the aperiodic event-activity network and the corresponding aperiodic timetable, a corresponding *vehicle schedule* assigns to each trip a vehicle with a certain capacity, specifying the maximum number of passengers who can use it simultaneously. The vehicle schedule implies the turnaround activities in the event-activity-network. Each trip needs exactly one incoming and one outgoing turnaround activity, connecting several trips to a vehicle tour. These vehicle tours may need to start and end at one or multiple given depots, depending on the dataset. Note that turnaround activities imply bounds as well, i.e., there are minimal turnaround times between two trips, including the time to drive from the end of the first trip to the start of the next one. A vehicle schedule is feasible if all trips are covered by a vehicle tour and all vehicle tours satisfy their turnaround activity bounds.

The aperiodic event-activity network is used as the basis for propagating delays and for finding the passengers' routes.

Propagating delays. Source delays are propagated along the activities in the network in the way common in delay management (Rückert et al., 2017; Dollevoet et al., 2018). Basically, if a vehicle starts some activity with a delay it also ends this activity with a delay. However, if the minimum execution time of some activities is shorter than the planned time, the available slack reduces the delay during propagation. For headway and transfer activities this depends also on the delay management strategy. In case a vehicle waits for a delayed connection, the delay propagates to the waiting vehicle. On the other hand, if the vehicle does not wait, there is no delay propagation. In our simulation we assume the delay management strategy as fixed. Note that our simulation framework supports dependency arcs between any arrival and departures arcs, hence waiting time rules for specific connections can be applied as well as a dynamic decision, e.g. based on Rückert et al. (2017). It is therefore possible to combine different delay management strategies with our approach. Depending on the context they can be tailored to the specifics of the public transport system, for example to urban bus systems including bus bunching, see, e.g. Fonzone et al. (2015), to metro systems or to long-distance trains.

Table 1
Robustness tests RT1–RT4 with a description and a motivation, as well as the parameters used in our experiments.

Name	Description	Motivation	Parameter for paper
RT-1	Initial delay of a single vehicle	Emulates problems at the beginning of a trip	Source delays of 5 minutes
RT-2	Slow-down of single network sections	Emulates problems like road work	Increase of travel time of section by 2 min
RT-3	Temporary blocking of single station	Emulates a gridlock at a station	Blocking of 15 min
RT-4	Random delay simulation	Emulates multiple common independent delays	Empirical distribution of delays based on Friedrich et al. (2018), see Fig. 2

Finding the passengers' routes. Inside the aperiodic event-activity network, we determine the passengers' routes using the given passenger demand data. For each passenger, we know the origin and destination of the desired journey, as well as the earliest departure time. Each passenger chooses the route that maximizes a utility function. The utility function can be customized and is here given by a weighted sum of the travel durations and the number of transfers used. We call this utility function the *perceived travel time*. As stated before, we assume for simplicity that the passenger demand given by origin–destination pairs is fixed, i.e., does not depend on the timetable.

For each vehicle its capacity is interpreted as the maximum capacity beyond that passengers cannot board anymore. To obtain realistic results, the passenger behavior regarding vehicle capacities needs to be modeled with some care. To do that, we first implemented a model, where passengers do not know the route choices of other passengers. In particular, they cannot base their own decisions on the available free vehicle capacity which in turn depends on the routes of the other passengers. Whenever several passengers want to board a vehicle, the available places are randomly distributed among them. Passengers may be forced to adapt their planned route on the fly when they realize that a vehicle cannot be boarded because of lack of free capacity or because they already know that they will miss a planned connection due to some delay.

Frequently adapting many passenger routes is computationally expensive. In order to save computation time we approximate the resulting passenger routes by using the following simplified, second model. Whenever a passenger needs an alternative route, we compute a fastest route to the destination which always respects the vehicle capacities subject to the planned routes of all other passengers. This model thus assumes a centralized routing algorithm knowing the desired paths of all passengers. Before using this model we tested that it generates (nearly) the same passenger delay distribution as before to be sure that it is a reasonably good approximation.

In recent years, many new speed-up techniques have been developed for finding passengers' routes efficiently, for a survey see Bast et al. (2016). Within the robustness tests, we selected the connection scan algorithm (CSA), a relatively easy to implement variant with excellent cache-efficiency which finds optimal paths within few milliseconds per route (Dibbelt et al., 2013, 2018).

2.2. Robustness tests

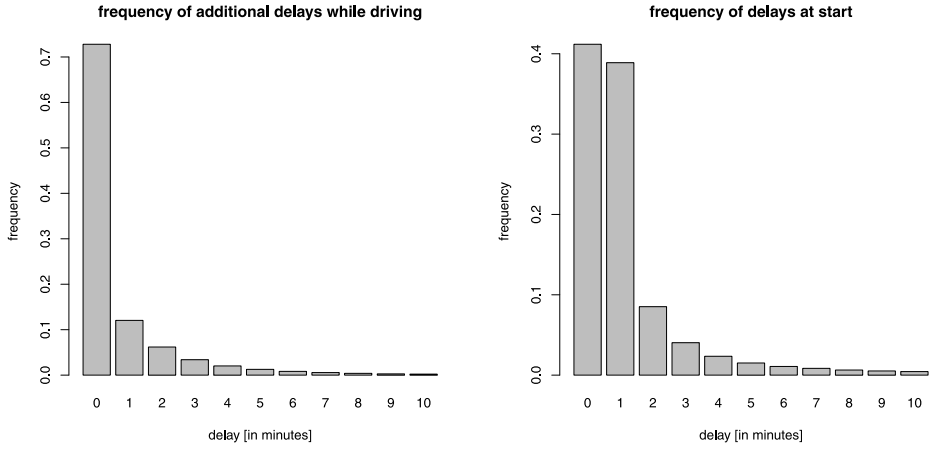
Friedrich et al. (2017, 2018) introduced several robustness tests for evaluating public transport systems on which this work is based. Here, the robustness of a public transport schedule is the simulated robustness against delays for a large number of plausible scenarios.

All robustness tests simulate certain aspects of common delays during daily operation, i.e., these tests create simulations that either execute specific stress tests or resemble empiric data on delays. Every test is a series of independent simulations measuring the sum of perceived delays of all passengers compared to their initially planned arrival time. We call this the *robustness value* of the simulation. We use the following four robustness tests and their corresponding evaluation from Friedrich et al. (2018). For a better understanding of how these tests work we give a detailed explanation of the first test as well as a short description of the other three.

The test RT-1 simulates a starting delay of any vehicle of the schedule. Starting delays for vehicles typically occur in daily operation. Therefore, the evaluation of RT-1 measures the robustness against this type of delay. Technically, we simulate a whole day separately for every single vehicle:

- We add a delay of x minutes at its start. The starting delay x is a parameter that can be set to a value best matching expected starting delays for the investigated network. If there is no knowledge about x, the test can also be used to iterate over all plausible values. For this paper, we used $x = 5$ as an example.
- As delay management strategy, we assume a no-wait policy, i.e., passengers miss their connecting trip in case they arrive too late. However, delays are propagated correctly through the network, i.e., existing slack is used to reduce the delay of the vehicles if possible. We further assume that no alternative transportation mode is provided to bridge interruptions.
- Recall that passengers may adapt their routes to the current situation in which delays occur.

For every single vehicle, the simulation then results in a sum of delays for all passengers in the network, called the robustness value for this specific simulation. We add up the robustness values for all simulations to obtain the robustness value of the robustness test RT-1 for this specific instance.



(a) Delay distribution for driving arcs

(b) Delay distribution for initial departure of trips

Fig. 2. Delay distribution for RT-4.

The other robustness tests are shown in Table 1. As for RT-1, we also assumed a no-wait delay management policy for them, and we also added up the values for the single simulations (over all network sections in RT-2, over all stations for RT-3 and over all randomly chosen delay scenarios in RT-4). Note that RT-1 to RT-3 are deterministic, and RT-4 is stochastic. Empirical delay distributions used in RT-4 for driving arcs and for the start delay of each trip are shown in Fig. 2. We also remark that our machine learning oracle can be trained with other robustness definitions apart from RT-1 to RT-4.

After conducting all robustness tests we compare the different instances as follows: Because all instances use the same passenger demand, we simply compare the robustness values of the different tests relative to one another. For better readability, we scale all robustness values such that the instance with the worst robustness value is scaled to the value 100. An instance producing only 10 percent of the robustness value of the worst instance gets therefore a value of 10 after scaling.

3. Machine learning of robustness

The goal of our work is to develop a predictor that is specific for each public transport network with a fixed line concept and passenger demand. We call this a *dataset*. Hence, we want to train a separate model in each case. For each dataset, a large number of timetables and vehicle schedules as well as corresponding passengers' routes are computed. Recall that these data together is called an instance. For each of the generated instances, we apply the robustness tests from Section 2.2 and obtain the corresponding robustness values.

The learning task is to predict robustness values for instances. Since the description of each instance is huge, we prefer to work with selected key features which work as a kind of fingerprint and characterize them. The importance of feature selection and feature engineering to the success of machine learning is well recognized, see e.g. Kuhn and Johnson (2019).

Choosing key features. The selection of key features representing each instance is challenging because several factors can influence different aspects of robustness. Building features that help to predict robustness in our instances need to serve two main goals. The first is representing information about features that are directly associated with robustness. These features contain the number of vehicles used per journey as well as the slack on activities (drive, wait, transfer, turnaround). The values of these features are a base for evaluating the robustness. The second goal of features is to build a context around them. For example, the amount of slack on transfers at a specific station, say station 5, is more useful with the context that there is a massive amount of transfers taking place at this station. During training, the predictors learn the important associations between features serving these two goals. In image processing, e.g., neural networks learn about the context of a 2D pixel even if the 2D image is converted into a 1D vector before processing. We selected a set of key features serving these two goals. Other considerations for the selection process were the following. The features should

- be numeric or labels that can be mapped to numerical values, see e.g. Raschka (2015),
- capture non-trivial aspects of the public transport system,
- be chosen so that common information or information that is likely to be similar between instances is under-represented while taking care to include differences, see e.g. Goodfellow et al. (2016).

The key features are grouped to ordered sets of fixed size, i.e., vectors whose values are extracted from each instance. In our model one instance produces nine vectors containing key features. These vectors contain information about the data in a network

Table 2
Key features of an instance.

#	Description	# elements
1	The avg. occupancy rate of the corr. vehicle in percent for each drive activity	m
2	The number of passenger groups with a perceived travel time of i minutes	$\text{traveltime}^{\max}$
3	The share of passengers with i transfers	$\#\text{transfers}^{\max}$
4	The average slack on wait activities per station	n
5	The average slack on transfer activities per station	n
6	The share of transfers happening per station	n
7	The average sum of line frequencies per station	n
8	The share of events happening per station	n
9	The number of trips with an outgoing turnaround slack of i minutes	$\text{turnaround}^{\max}$

with n stations and m network edges. Table 2 shows a description of these feature vectors. Note that for the vectors with an upper limit, e.g., the travel time vector, values larger than the upper bound are counted as the upper bound, e.g., a passenger with a travel time of $\text{traveltime}^{\max}+1$ min is counted in the last vector entry. All nine vectors are joined to one large vector as input for the machine learning procedure.

A different approach to this problem would be to use all available information and learn from the schedules directly. Apart from the technical problem that different schedules have in general different sizes, this would, however, require a large scale deep learning approach that may become more susceptible to over-fitting. We would also forfeit the benefits mentioned above.

Choosing suitable ML models. The task for the machine learning algorithm is to use the set of key features and produce multiple values estimating the outcome of the robustness experiments. We suspect that the relationship between input and output is highly non-linear and to train the algorithm we are using supervised learning. The methods of choice for this situation are artificial neural networks (ANN) and regression-based methods like support vector regression (SVR), or gradient boosting (Goodfellow et al., 2016). A priori, it is not clear which of the regression models will be best suited for our application. In Section 5, we will present the evaluation of many common models, used in similar regression scenarios (Mahdavinejad et al., 2018; Fan et al., 2019). For all of those techniques, we used Python and open-source libraries. The data and scripts for our oracle generation are publicly available so that the parameters and libraries used are transparent, see Müller-Hannemann et al. (2022).

Designing an artificial neural network (ANN) requires a more extensive process than the other methods used. Artificial neural networks can have different structures and number of nodes. We use a model with a fixed number of layers and neurons. Every neuron of one layer is connected to every neuron of the previous layer. This is often called a *feedforward neural network* (Schmidhuber, 2015). If this network has multiple layers it is usually called a *multi-layer perceptron*.

4. Application: A local search framework

In the following we demonstrate the usefulness of a machine learning model at a first application in which we introduce a simple local search algorithm to improve the robustness of a given public transport schedule by adapting its timetable. The goal is to add additional slack at strategic points in the network while still maintaining a good quality for the passengers, given by the sum of utilities which in our case describe the perceived traveling times.

The algorithm is sketched in the right part of Fig. 1 and presented with pseudocode in Algorithm 1. The algorithm itself is published as part of the open-source software library LinTim, see Schiewe et al. (2022, 2021). It starts with an instance as a starting solution, i.e., some infrastructure and passengers' demand, a line concept, an aperiodic timetable, a vehicle schedule, and corresponding passengers' routes. As mentioned before, the infrastructure, the passengers' demand and the line concept is assumed to be fixed. We now try to improve the timetable, the vehicle schedule and the corresponding passengers' routes. The combination of an aperiodic timetable, a vehicle schedule and passengers' paths is called a *solution*. To this end, the algorithm generates a neighborhood for the current solution and evaluates the robustness of all elements of the neighborhood by using the oracle. The best element is chosen as the next solution if its utility is not too bad.

For a given instance we define a neighborhood of $4N$ elements as follows: We first find the most promising drive, wait, change and turnaround activities by choosing N activities with the smallest current slack for each activity type. For drive, wait and change activities the slack is divided by the number of passengers as an additional weight factor. We obtain activities $a_1, \dots, a_{4N}$.

For each of the chosen activities, $a_1, \dots, a_{4N}$ we proceed as follows. We increase the slack of $a_k = (i_k, j_k)$, resulting in a later time for event j_k. This new aperiodic timetable may be infeasible, namely if any of the resulting durations of the activities which start at j_k is smaller than its lower bound. In this case, also the times of the subsequent events have to be increased until the lower bounds of the respective activities are met. This is continued until the increase of the slack time of a_k is compensated by the sum of slack times of subsequent activities. Note that the upper bounds of the activities are not checked here, we assume that all events can be postponed arbitrarily.

Afterwards, the new public transport schedule is evaluated using the robustness oracle and the best solution in the neighborhood is implemented. Note that we only choose a new solution in our implementation if the passenger utility does not decrease too much in comparison with the start solution. For this, a limit for the amount of passenger utility decrease can be specified by the user, e.g., 10%. For a more thorough investigation about using the oracle within local search algorithms we refer the reader to Müller-Hannemann et al. (2021).

Algorithm 1: Local search using machine learning

```
Data: the starting solution currentSolution
currentValue = evaluateByOracle(currentSolution)
while true do
    bestImprovement = ∅
    bestValue = ∞
    foundImprovement = False
    Compute local neighborhood of currentSolution
    if Rerouting step? then
        Reroute all passengers and update currentSolution
        currentValue = evaluateByOracle(currentSolution)
    end
    for newSolution in local neighboorhood do
        introduceAdditionalSlack(newSolution)
        value = evaluateByOracle(newSolution)
        if passengerUtility(newSolution) too bad then
            continue
        end
        if value < bestValue then
            bestValue = value
            bestImprovement = newSolution
        end
    end
    if currentValue > bestValue then
        currentValue = bestValue
        currentSolution = bestImprovement
        foundImprovement = true
    end
    if not foundImprovement then
        break
    end
end
```

For a correct evaluation, a rerouting of the passengers is necessary in every step in order to deliver the correct key features to *oracle* and in order to correctly compute *passengerUtility*. However, to save computation time, the rerouting step is only included every few iterations (in our experiments, we reroute every 10 iterations). In between the rerouting steps, the passengers' routes are assumed to be constant even though the timetable changes.

Note that Algorithm 1 sets a general framework for using machine learning in a local search for robust solutions. Many variations are possible: the definition of the neighborhood may be refined in order to improve the local search. Moreover, the oracle can be chosen according to the planner's requirements such that the framework is also usable for other settings as the one described in this paper.

5. Experiments

To evaluate our approach, we are considering four datasets: The artificial benchmark datasets `grid` and `ring` (see FOR 2083, 2018), the bus system in Göttingen, Germany (`goevb`) and the regional train network in southern Lower Saxony, Germany (`lowersaxony`). All datasets can be found as part of the open-source software library LinTim, see Schiewe et al. (2022, 2021). The dataset contains the infrastructure as well as the passenger demand directly as parameters while the line concept is generated by LinTim using the basic cost-oriented approach of Schöbel (2012). For an overview of the dataset sizes and figures of the infrastructure network, see Appendix. Note that we considered urban bus networks as well as rail networks, allowing us to investigate the usefulness of our approach for different settings, e.g., for high and low line frequencies. For this, parameters such as the vehicle capacities or the transfer weight in the perceived travel time were adapted accordingly.

To find the initial public transport schedules used to train our model, we used state-of-the-art methods from mathematical public transport planning implemented in LinTim to compute several thousand instances for each dataset. As timetabling methods we used solution procedures for the classic PESP model (as usual in periodic timetabling), in particular a cycle based integer programming (IP) formulation (Peeters and Kroon, 2001) as exact solution method and the fast MATCH heuristic (Pätzold and Schöbel, 2016), coupled with different strategies for distributing slack times such as adding e.g. an exponential slack distribution on all activities or a proportional slack on the wait activities in the most used stations. Vehicle scheduling was done by solving the standard IP flow formulation (Bunte and Kliewer, 2009) which minimizes a weighted sum for the operational costs, and using several adaptions such

Table 3
This table shows basic parameters of the studied infrastructure networks ($|S|$ denotes the number of stations), the number of instances created, and the resulting quality of our robustness predictions for SVR (sd denotes the standard deviation). The mean relative error when predicting the robustness of an instance is always below one percent. The last two columns show the percentage of cases with at least 99% and 95% accuracy. Hence, outliers are also relatively rare.

Name	Network type	$\|S\|$	|instances|	Error		≥ 99% accurate	≥ 95% accurate
				Mean	sd		
`grid`	artificial	80	8304	0.72	0.85	75%	99%
`ring`	artificial	161	2768	0.29	0.87	95%	96%
`goevb`	real world	257	4152	0.86	1.30	75%	98%
`lowersaxony`	real world	35	5536	0.36	0.77	90%	99%

as fixing possible vehicle schedules before the timetabling step (Pätzold et al., 2017) or different values for the minimal turnaround times of vehicles. To allow reproducibility, all instances created for training along with their corresponding LinTim parameters used for the creation are published, see Müller-Hannemann et al. (2022).

Afterwards, these instances were evaluated using the robustness tests described in Section 2.2. We use these tests to obtain the values to learn from the key features presented in Section 3. In our experiments the maximal values were 240 min for the maximal travel time $\text{traveltime}^{\max}$, 10 for the maximal number of transfers $\#\text{transfers}^{\max}$and 30 min for the maximal turnaround time $\text{turnaround}^{\max}$. Note that these values are highly dependent on the type of datasets used, e.g., the maximal travel time probably needs to be higher when considering long-distance railway networks. All robustness test results and their corresponding key features can be found at Müller-Hannemann et al. (2022).

In the following, we first present the experimental results regarding the developed machine learning predictors in Section 5.1, before investigating the performance of the local search procedure in Section 5.2.

5.1. Machine learning results

To measure the quality of estimation for one robustness test we use the average error and the standard deviation between the real robustness and the estimated robustness measured in values as explained in Section 2.2. Since we have four robustness tests, this produces four error rates. In Table 3 we display the mean error as the average of the four values.

Which machine learning model to choose?. We tested several machine learning methods for estimating the robustness values. The libraries we used for the ANN are NumPy (Oliphant, 2016), a python library for handling large multidimensional arrays and matrices, TensorFlow (Google Brain Team, 2015), a library for machine learning where computations are expressed as stateful dataflow graphs, and Keras (Chollet, 2015), a python library for artificial neural networks that acts as an interface for TensorFlow. Additionally, we used several models that are part of the scikit-learn open-source library (Pedregosa et al., 2011). More specifically, we used Logistic Regression, Decision Tree Regression, Gradient Boosting Regression, Bayesian Ridge Regression, Elastic Net Regression, and Support Vector Regression. In addition to scikit-learn we used CatBoost (Prokhorenkova et al., 2019) and XGBRegressor (Chen and Guestrin, 2016).

For each of these models, one or more hyperparameters have to be optimized to achieve their best accuracy. In principle, one could optimize the hyperparameters of each model independently for each of our for datasets and for each of the four robustness tests, possibly yielding 16 different sets of parameters for each model. We refrained from this and kept parameter tuning to a minimum. Unless otherwise stated, we kept the default hyperparameter configuration for the respective method. A notable exception is the support vector regression (SVR) where we experimented with different kernel functions. The best results are obtained with the radial basis function (RBF), while the polynomial kernel with degree three turns out to be the second-best kernel.

We also investigated more work into the artificial neural networks. It has to be specified how many hidden layers the network has, what type of connection the layers have, and how many neurons are in each layer. After testing several configurations, we find that the combination of 150 neurons per hidden layer and five hidden layers that are fully connected yield the best results.

The quality of the prediction in all our models is evaluated using a separation of the data into training- and testing sets in an 4:1 ratio. For ANN, we chose a separation into training-, validation and testing sets in an 8:1:1 ratio. For the latter, the training operates in many small runs, called epochs, and in each epoch, the network trains on a subset of the whole training data and is evaluated with the validation set to prevent over-fitting.

Fig. 3 shows the performance of all tested models. Models with a relatively large mean error are Bayesian ridge regression (mean error 1.75) and elastic net regression (mean error 2.435). These two models both belong to a family of linear models which may explain their poorer performance in comparison to other models. For the ANN, we tried two approaches, one with four distinct models for each robustness test only (ANN four nets) and one integrated model with four output neurons (ANN one net). Interestingly, the integrated approach with four output neurons performed better on the test set. Support Vector Regression (SVR) yields the smallest mean error of all models tested. However, XGBoost comes very close while using fewer parameters.

Based on these results, any of the models XGBoost, Catboost or SVR would be viable to resume with, since those models are non-dominated w.r.t. their mean and standard deviation error values presented in Fig. 3. Therefore, we applied an additional test, testing the different models on solutions generated by the local search procedure described in Section 4, i.e., on instances with different structures not observed in the training data. There, we observed the best results for SVR, which is the reason why this will

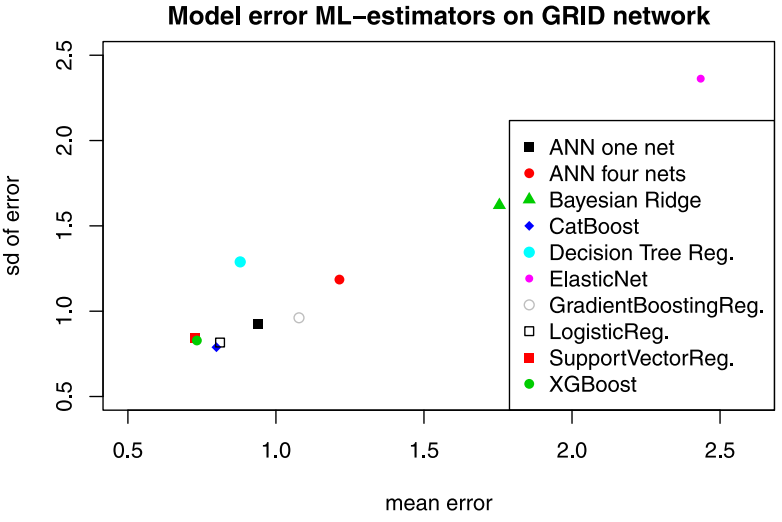


Fig. 3. Comparison of multiple regression models to estimate the results of all four robustness tests. Here we look at the mean error and standard deviation for the dataset `grid`.

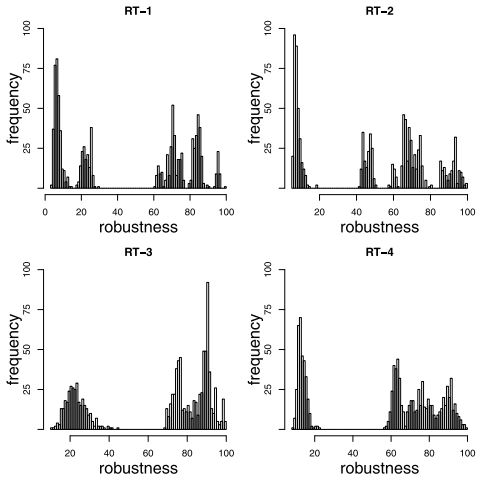


Fig. 4. Histogram showing the distribution of (real) robustness values over all test instances for the `grid` network.

be the model investigated for the remainder of this paper. As mentioned earlier, these results can be further improved by thorough hyperparameter optimization. But for the practical application as a robustness oracle, SVR with its current error rate satisfies our expectations.

Evaluation of the quality of our predictions. One of the central achievements of this paper is a good predictor for our four robustness measures. The evaluation of our machine learned oracles therefore has four distinct error rates. These four errors are similar (see Fig. 5), so we provide the average error over all tests. We present our findings in Table 3.

To better understand the nature of the results we show the distribution of robustness values for robustness tests RT-1–RT-4 for the `grid` network in Fig. 4. We observe that instances are clearly clustered around a few peak values and that there are large gaps in all four robustness tests. The latter means that our sampling of instances has never produced instances with such values. These clusters correspond to the different timetabling and vehicle scheduling strategies used, e.g., solutions with additional slack added always have better robustness. For more details on how different strategies affect the robustness of instances in the tests used here, compare (Friedrich et al., 2017, 2018).

Fig. 5 shows the deviation of predicted robustness values (*estimated robustness*) from exactly computed robustness values (*real robustness*) for each of the four robustness tests. Overall we observe an excellent general agreement of real and estimated robustness

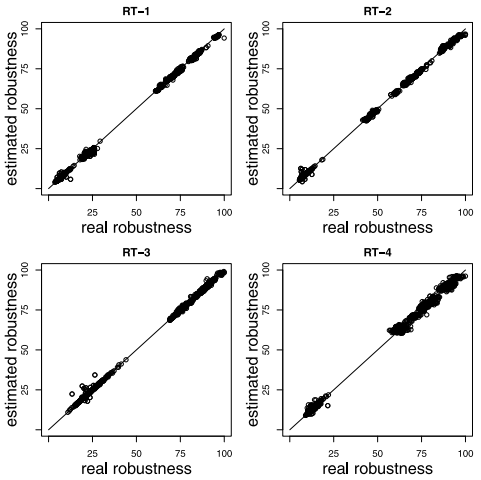
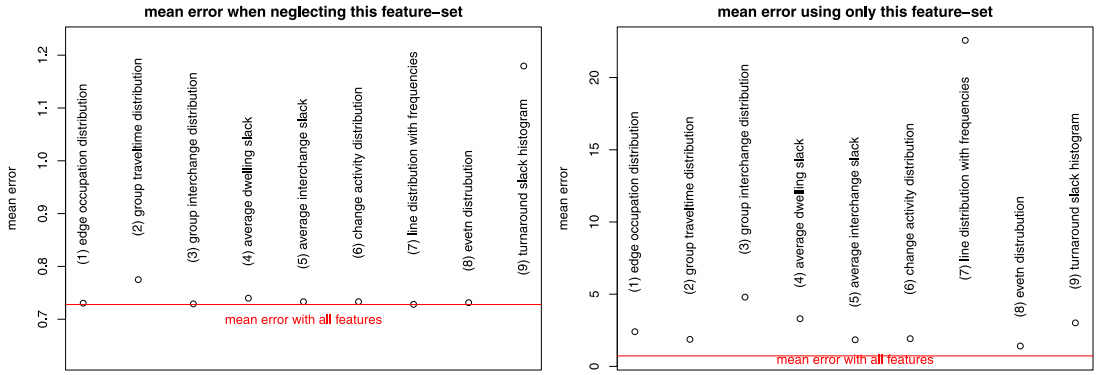


Fig. 5. SVR predictions of all four robustness tests for all instances of the `grid` network. Real robustness refers to the result of a complete simulation.



(a) Performance of estimation when one feature set is not used (grid network)

(b) Performance of estimation when only one feature set is used (grid network)

Fig. 6. Analysis of the importance of single key features for SVR.

values with relatively small deviations and few outliers. The numbers in Table 3 show that our network produces results of sufficient accuracy to be useful in our framework. The predictions produced by the oracle have a mean error that is below one unit of robustness in percent. Observe that robustness test RT-4 has significantly more deviation than robustness tests RT-1–RT-3, due to its nature of being a stochastic test. Overall, we observe only a small number of outliers. Some of them can be seen in Fig. 5 for RT-3 in the lower left corner.

The oracle may not be able to predict values accurately that are well outside those seen so far. For example, a new instance for `grid` may have a real robustness value of 40 for RT-4 but the estimation may not be able to produce a value between 25 and 50 with high accuracy. An example for this effect can be seen later for an instance of `goevb`, compare Fig. 10(a).

A critical analysis of our key features. In Section 3 we introduce nine sets of key features containing multiple values for each set. We conducted two experiments to assess their contribution to the accuracy of the robustness estimation based on SVR. These experiments deliberately contain feature set seven *(the average sum of line frequencies per station)*, although the lines and their frequencies are fixed in all of our experiments. We designed this feature set with the intention of using it in the future, when the line concept may not be fixed any more. In this experiment, it serves as a control feature that shows how an unnecessary feature will behave.

In the first experiment, we trained our SVR model with all but one feature sets. The results of tests in which one feature set was omitted in each case are shown in Fig. 6(a). The interesting finding is that only neglecting feature set two (the perceived travel time distribution of passengers) and feature set nine (the distribution of the available turnaround slack between subsequent trips)

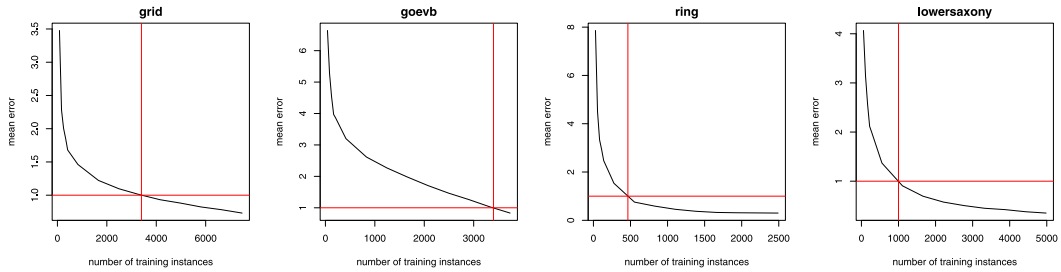


Fig. 7. Number of instances necessary until the average error converges for the SVR model.

Table 4
Runtimes of typical local search runs for each dataset, comparing prediction (pred.) times with corresponding simulation (sim.) times. The prediction time is given as an average over all predictions.

Dataset	# pred.	pred. time	sim. time	Runtime with pred.	Runtime with sim.	Speed-up factor
`grid`	1780	47 ms	203 s	611 s	362,313 s	593
`ring`	520	15 ms	500 s	232 s	260,284 s	1122
`goevb`	900	46 ms	1189 s	1499 s	1,071,537 s	715
`lowersaxony`	1420	16 ms	641 s	353 s	910,949 s	2581

result in a significant performance loss. As expected, feature set seven (the distribution of line frequencies) does not influence the result. Neglecting other features results in a very small loss of accuracy. One explanation for this behavior is that there might be some kind of redundancy between the other feature sets.

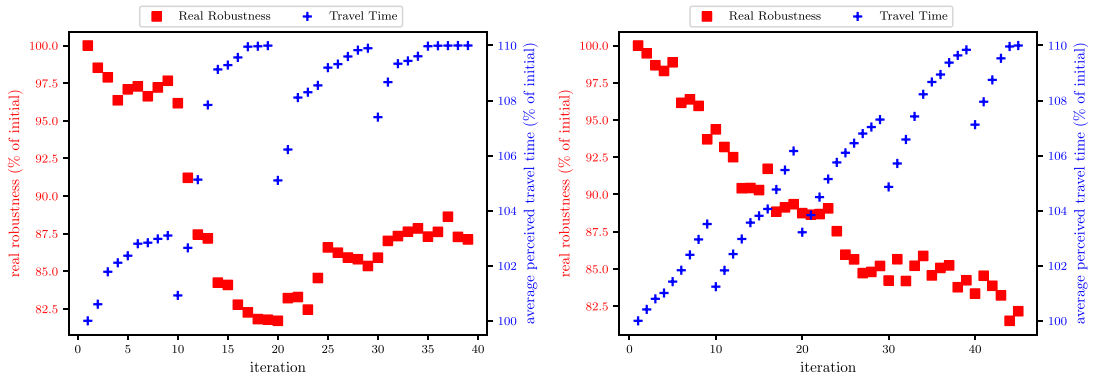
In the second experiment, we trained our SVR model using only single feature sets, see Fig. 6(b). The result is very different from the other experiment. Feature set nine, for example, is relatively bad for estimating the robustness by itself while neglecting this feature resulted in a significant loss of accuracy. Surprisingly, feature set eight (the distribution of events to stations) is most accurate in predicting the robustness if taken alone, although it does not contain any information about slacks, occupation, or travel time. This feature set seems to act like a fingerprint of the structure of the instance. It can be used for comparing the similarity of instances. We conclude that feature sets two and nine are essential for a good estimation. Other features (except feature 7) only yield small benefits in this setting. An evaluation of new instances created in Section 5.2 revealed that single feature models result in larger errors in the estimation. Features set eight alone, for example, produced mean errors twice to five times larger compared to the complete model.

How many instances are needed to achieve a good quality? In this experiment, we limited the number of instances available for training and tested the performance prediction for the robustness with the remaining instances. Fig. 7 shows how fast the average error rate drops with an increasing number of training instances. For the instances of the `grid` and `goevb`, we observe that more than 3400 instances are necessary to achieve an average error below 1%, thus for a reliable prediction using our set of key features. The other two smaller networks showed similar behavior, but only needed 500 and 1000 instances. This shows the high initial cost of our proposed machine learning approach which needs to be compensated by a time benefit for later invocations of the model to justify our approach.

How much time do we invest and how much do we save? One of the central motivations for this work is to save computation time in an iterative optimization process for robust public transport schedules. In this section, we evaluate how expensive robustness tests are and how much time can be saved using our technique. To compare computation times between the simulation-based approach and the machine learning-based approach, several factors need to be considered.

First of all, to allow a machine learning-based approach, a large number of instances need to be precomputed to allow the training of one of the machine learning predictors described above. As can be seen in the paragraph above, the predictors improve when trained with several thousand instances in the training set, leading to high startup costs for this approach. We therefore need several hours per dataset to create and evaluate the training data, depending on the predictor accuracy that should be achieved. Compared to that, the training process itself is quite fast, requiring at most a few minutes.

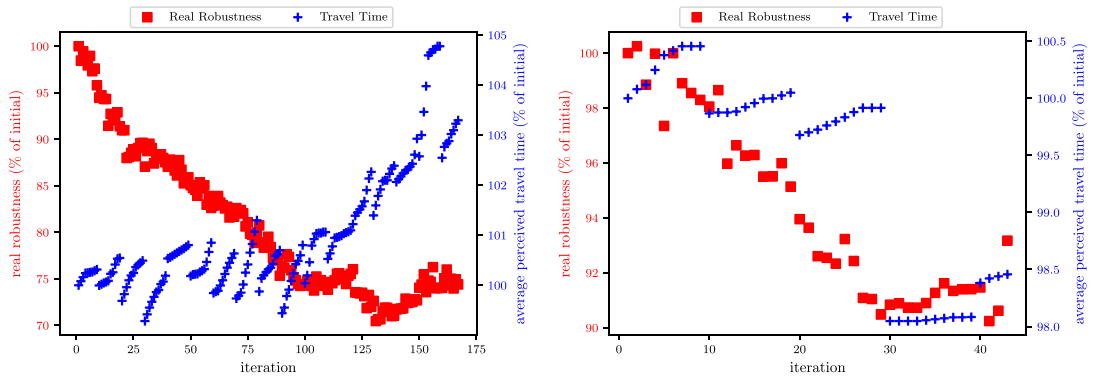
On the other hand, the evaluation of an instance is much faster when such a machine learning predictor was trained beforehand. Table 4 gives an overview of a typical local search procedure for one instance per dataset. Since the number of computed predictions is high (520–1780, depending on the instance), the saved time by using a machine learning-based predictor is large. While the average prediction time is between 15 and 46 ms, the average simulation time for all four robustness tests may take up until 20 min. A fair comparison of both variants has to include also all other parts of the algorithm. The fifth column of Table 4 presents the overall runtime when using oracle-based robustness predictions, while the sixth column presents the overall runtime with simulations. The



(a) Example instance of goevb

(b) Example instance for lowersaxony

Fig. 8. Solutions with a bad starting robustness.



(a) Example instance for grid

(b) Example instance for ring

Fig. 9. Solutions with a very robust starting robustness.

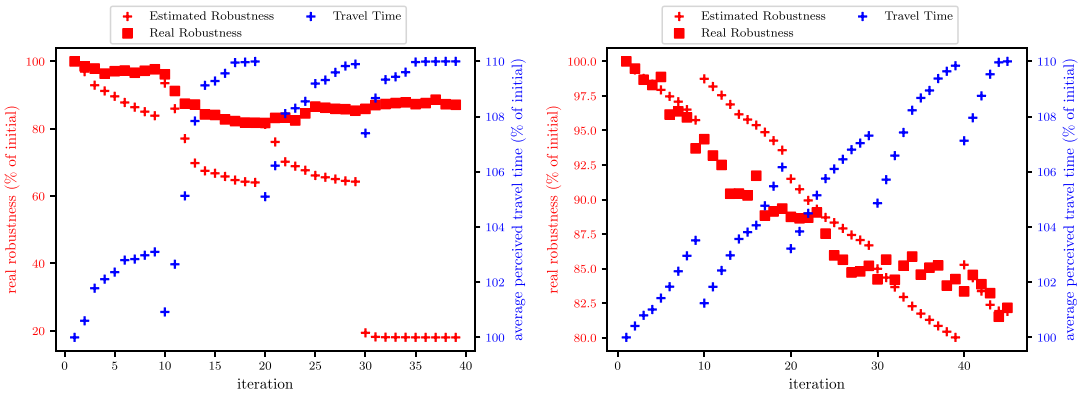
final column gives approximate speed-up factors which we achieve when using the oracle instead of the simulation. These speed-up factors range between 593 and 2581, showing that we obtain speed-ups by two and three orders of magnitude.

Note that the values for the runtime with the simulation are only an approximation, since we did not run this experiment due to the immense time needed but used the time for a typical simulation (without passenger routing since this already takes place in the local search) to approximate the overall time needed. Of course, using the exact simulation may change the number of predictions needed (in either direction).

5.2. Local search results

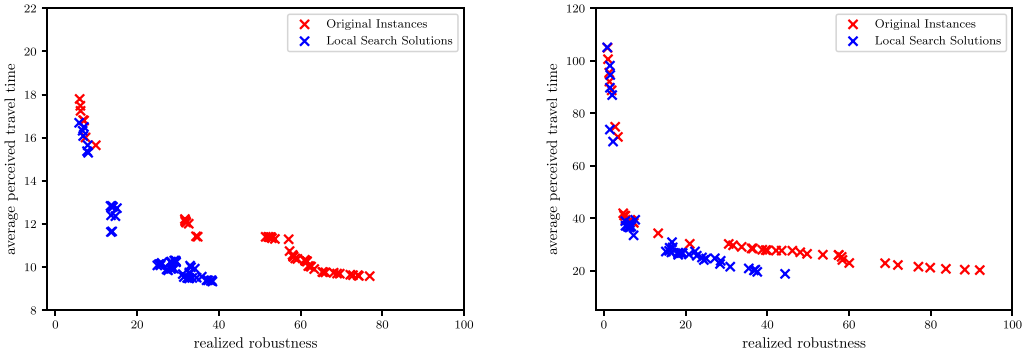
So far we have established the quality of our estimation when predicting the robustness of instances generated through LinTim's central process. With the following experiments we address two goals. We present the first results of a local optimization framework using the machine learning predictions as an oracle. We then can proceed with a different kind of validation, testing instances generated by this process. For the evaluation, we selected example instances to represent the benefits and possible shortcomings of this approach and chose a local neighborhood size of $4N = 80$, a rerouting interval of 10 and a possible travel time increase of 10%.

A typical instance to represent the benefits of the approach has a relatively good cost and connections for the passenger but suffers from bad robustness. Fig. 8(a) shows the behavior of such a starting solution for the local search on the dataset `goevb`. After 40 iterations, the robustness of the instance is improved by 13%, while at the same time the average perceived travel time is increased by 10% in its trade-off which causes the local search to terminate. Similarly, for dataset `lowersaxony`, Fig. 8(b) shows an improvement in robustness of 17.5% while increasing the average perceived travel time again by 10%. Note that we are showing the real robustness here, i.e., the robustness given by simulations of the corresponding instances instead of the predicted



(a) goevb (b) lowersaxony

Fig. 10. Measuring the gap between estimated and real robustness.



(a) grid (b) ring

Fig. 11. Comparing travel time and robustness for multiple input instances. The local search procedure was used on all non-dominated instances in the training set.

value. Furthermore, we see that while the local search only chooses a new solution if the predicted robustness is better than before this must not be the case for the real robustness, i.e., the real robustness observed is not monotone. Another aspect to note is the jumps in the perceived traveling time every 10 steps. Both of these effects are (partly) caused by the chosen rerouting interval of 10, resulting in a rerouting of all passengers every 10 iterations. In between, the passenger paths are assumed to be constant even when the public transport schedule changes, compare Algorithm 1. This results in a small error when estimating the perceived travel time and key features but improves the runtime of the local search.

Figs. 9(a) (for dataset `grid`) and 9(b) (for dataset `ring`) are instances starting with an already good robustness. But both instances can still be improved with respect to robustness, by about 25% and 7%, respectively. Interestingly, the improvement in robustness for the `ring` instance is achieved while improving the average travel time for the passengers, providing a solution that is more robust and improves the perceived travel times for the passengers, too.

But one disadvantage of the approach proposed here is, that the local search is only using the machine learning predictor to assess the solution quality and has no real knowledge about the robustness. To investigate this effect further, we revisit the example from Fig. 8(a) and observe not only the reevaluated real robustness but the predicted robustness as well in Fig. 10(a). Here we see, that although the real robustness is improved by 13%, the estimated robustness improvement is much larger with about 80%. Therefore, the local search procedure is overestimating the robustness improvement immensely. It is therefore not advisable to rely on the local search alone to assess the solution quality obtained. Results should in the end always be reevaluated using more extensive robustness tests. Note that this effect can only be seen in a few of our experiments, while the tracking of the real robustness is normally much better. An example for that is shown in Fig. 10(b), a revisit of Fig. 8(b). Additionally, note that this effect could be mitigated when solutions computed by the local search are used as additional training data, providing the machine learning with new structures to learn not present in the original training instances. But this requires a feedback loop from the local search back into the machine learning model that has not yet been integrated into this work.

Finally, we examine the solution quality of the local search solutions compared to the input instances. An example for that is Fig. 11, where the non-dominated instances of the training dataset were used as starting instances of the local search procedure. With this, the set of non-dominated solutions can be improved significantly. Especially for `grid` we are able to produce solutions that are far better w.r.t. the travel time of the passengers and the robustness.

6. Conclusions and future work

In this paper, we used machine learning to approximate the robustness of a public transport schedule. Since the evaluation of robustness means to simulate a large set of scenarios this saves significant amounts of computation time. Our machine learning model results in an oracle that is able to predict the robustness very accurately. In order to train the regression models we proposed key features which describe the characteristic features of the instances. The oracle may be used to improve the robustness of a timetable within a local search framework.

In a few instances generated within the local search procedure the predictions of the oracle did not match the real robustness values. The reason is structural differences in the instances in the training set. We hence work on new ways to generate training data covering also these cases.

We point out that the evaluation whether a timetable is robust depends on the delay management strategy. If e.g., a simple no-wait policy is chosen, a robust timetable will have more slack times on transfer activities as a robust timetable in an always-wait strategy. The delay management strategy is included in the robustness tests used in our approach and hence reflected in the resulting oracle. This opens the possibility to analyze how much robustness depends on the delay management strategy and how timetables that are robust without knowing the strategy look like.

Three other lines of work for the future should be pointed out. First, we work on including more aspects of public transport planning into the machine learning model, e.g., the line concept of an instance. Currently, we concentrated on the timetable and vehicle schedule while the underlying line concept is fixed. But also the line concept affects the robustness as shown in Goerigk et al. (2013b) and Schöbel and Schwarze (2013). Hence, including the line concept in the robustness evaluation gives more insight and may allow an even more accurate machine learning model and the design of robust line plans and line concepts. It is also interesting to evaluate whether the robustness of a public transport system is influenced more by the line concept or more by the timetable. Another planning aspect to include would be passenger demand instability, i.e., the influence of the timetable and its robustness on the modal split and therefore on the passenger demand which in turn is an input of the model.

Another important future step is the work on improved robustness optimization algorithms. While we were able to provide a proof of concept on how to use a machine learned oracle in a local search algorithm, developing future algorithmic approaches that use the oracle (or similar models) as a black box for robustness evaluation is a very interesting topic of research. This may include improved local search algorithms but also metaheuristics such as hill climbing, simulated annealing, or taboo search. In subsequent work, we have already demonstrated that a genetic algorithm leads to considerable improvements (Müller-Hannemann et al., 2021). Apart from finding robust timetables it may also be interesting to use similar techniques to create an oracle for subparts of the solution, e.g., to find robust routes for the passengers. This may yield a new approach for robust timetable information as defined in Goerigk et al. (2014) and Goerigk et al. (2013a).

Finally, a promising extension of our approach could be to predict not only the overall robustness of the public transport schedule, but additionally to locate where particular weaknesses occur.

CRediT authorship contribution statement

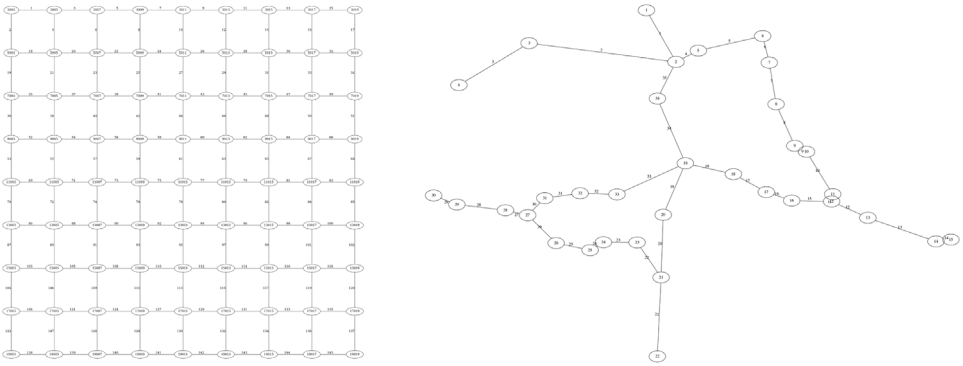
Matthias Müller-Hannemann: Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision, Project administration, Funding acquisition. **Ralf Rückert:** Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Alexander Schiewe:** Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Anita Schöbel:** Conceptualization, Methodology, Writing – review & editing, Supervision, Project administration, Funding acquisition.

Appendix. Datasets

See Table 5 and Figs. 12–14.

Table 5
Sizes of the used datasets.

Name	# Stations	# Edges	# Passengers	# Lines	# Events
`grid`	80	145	1676	30	728
`ring`	161	320	2022	37	1376
`goevb`	257	548	1943	22	2348
`lowersaxony`	35	36	11967	7	508



(a) Infrastructure network of grid

(b) Infrastructure network of lowersaxony

Fig. 12. Infrastructure network of `grid` and `lowersaxony`.

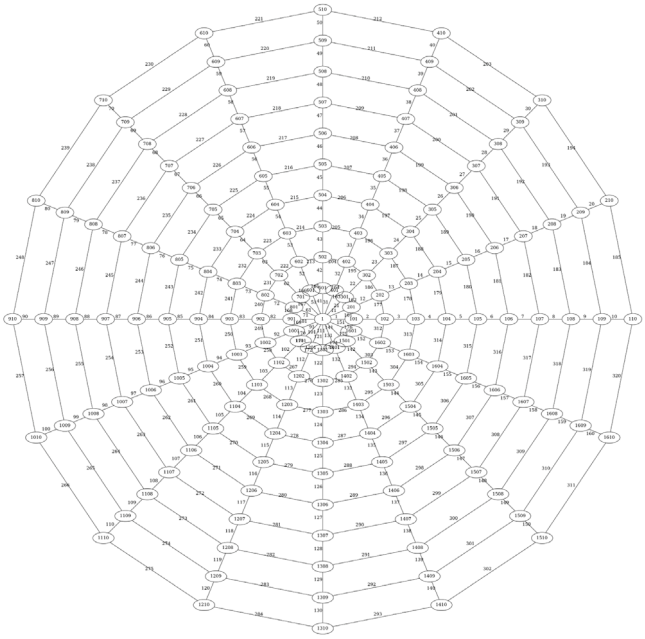


Fig. 13. Infrastructure network of `ring`.

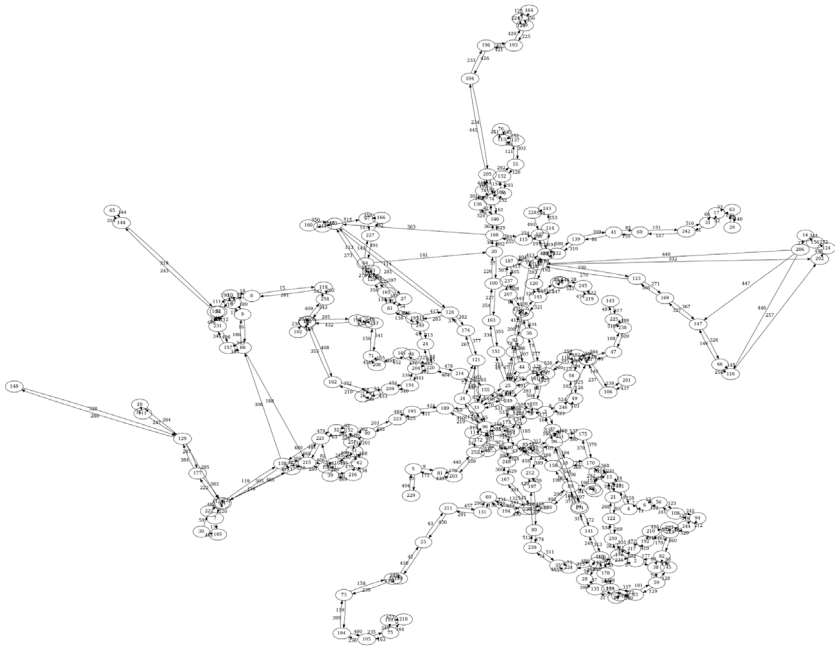


Fig. 14. Infrastructure network of goevb.

References

Andersson, A.V., Peterson, A., Törnquist-Krasemann, J., 2013. Quantifying railway timetable robustness in critical points. J. Rail Transp. Plan. Manag. 3 (3), 95–110.

Bast, H., Delling, D., Goldberg, A.V., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., Werneck, R.F., 2016. Route planning in transportation networks. In: Kliemann, Lasse, Sanders, Peter (Eds.), Algorithm Engineering - Selected Results and Surveys. In: volume 9220 of Lecture Notes in Computer Science, Springer, pp. 19–80. http://dx.doi.org/10.1007/978-3-319-49487-6_2.

Bauer, R., Schöbel, A., 2014. Rules of thumb — practical online strategies for delay management. Public Transp. 6 (1), 85–105, http://num.math.uni-goettingen.de/preprints/files/2011-03.pdf.

Borndörfer, R., Karbstein, M., Liebchen, C., Lindner, N., 2018. A simple way to compute the number of vehicles that are required to operate a periodic timetable.In: R. Borndörfer and S. Storandt (Ed.), 18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018). volume 65 of OpenAccess Series in Informatics (OASIcs). 16:1–16:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.https://doi.org/10.4230/OASIcs.ATMOS.2018.16.

Borndörfer, R., Klug, T., Lamorgese, L., Mannino, C., Reuther, M., Schlechte, T., 2018b. Handbook of Optimization in the Railway Industry, Volume 268. Springer.

Bunte, S., Kliewer, N., 2009. An overview on vehicle scheduling models. Public Transp. 1 (4), 299–317.

Bussieck, M., Kreuzer, P., Zimmermann, U., 1997. Optimal lines for railway systems. Vol. 96, no. 1, pp. 54–63.

Cacchiani, V., Caprara, A., Toth, P., 2010. Non-cyclic train timetabling and comparability graphs. Vol. 38, no. 3, pp. 179–184.

Cats, O., 2016. The robustness value of public transport development plans. J. Transp. Geograp. 51, 236–246.

Cats, O., Jenelius, E., 2018. Beyond a complete failure: the impact of partial capacity degradation on public transport network vulnerability. Transp. B: Transp. Dyn. 6 (2), 77–96.

Cats, O., Yap, M., van Oort, N., 2016. Exposing the role of exposure: Public transport network risk analysis. Transp. Res. Part A: Policy Pract. 88, 1–14. http://dx.doi.org/10.1016/j.tra.2016.03.015.

Chen, T., Guestrin, C., 2016. XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. In: KDD '16, ACM, New York, NY, USA, pp. 785–794. http://dx.doi.org/10.1145/2939672.2939785, http://doi.acm.org/10.1145/2939672.2939785.

Chollet, F., 2015. Keras - simple. flexible. powerful. https://keras.io/.

Claessens, M., van Dijk, N., Zwaneveld, P., 1998. Cost optimal allocation of rail passenger lines. Vol. 110, no. 3, pp. 474–489.

D'Angelo, G., Di Stefano, G., Navarra, A., Pinotti, C.M., 2009. Recoverable robust timetables on trees. In: Combinatorial Optimization and Applications. In: volume 5573 of Lecture Note in Computer Science, Springer, pp. 451–462.

De-Los-Santos, A., Laporte, G., Mesa, J.A., Perea, F., 2012. Evaluating passenger robustness in a rail transit network. Transp. Res. C 20 (1), 34–46. http://dx.doi.org/10.1016/j.trc.2010.09.002, Special issue on Optimization in Public Transport+ISTT2011.

Dibbelt, J., Pajor, T., Strasser, B., Wagner, D., 2013. Intriguingly simple and fast transit routing. In: Bonifaci, Vincenzo, Demetrescu, Camil, Marchetti-Spaccamela, Alberto (Eds.), Experimental Algorithms, SEA 2013. In: volume 7933 of Lecture Notes in Computer Science, Springer, pp. 43–54. http://dx.doi.org/10.1007/978-3-642-38527-8_6.

Dibbelt, J., Pajor, T., Strasser, B., Wagner, D., 2018. Connection scan algorithm. ACM J. Exper. Algorithmics 23 (1.7:1–1.7), 56. http://dx.doi.org/10.1145/3274661.

Dollevoet, T., Huisman, D., Schmidt, M., Schöbel, A., 2018. Delay propagation and delay management in transportation networks. In: Handbook of Optimization in the Railway Industry. Springer, pp. 285–317.

Fan, J., Wu, L., Zhang, F., Cai, H., Zeng, W., Wang, X., Zou, H., 2019. Empirical and machine learning models for predicting daily global solar radiation from sunshine duration: A review and case study in china. Renew. Sustain. Energy Rev. 100, 186–212. http://dx.doi.org/10.1016/j.rser.2018.10.018.

Fischetti, M., Monaci, M, 2009. Light robustness. In: Ahuja, R.K., Möhring, R.H., Zaroliagis, C.D. (Eds.), Robust and online large-scale optimization. In: Volume 5868 of Lecture Notes on Computer Science, Springer, pp. 61–84.

Fonzone, A., Schmöcker, J.-D., Liu, R., 2015. A model of bus bunching under reliability-based passenger arrival patterns. Transp. Res. Proc. 7, 276–299.

FOR 2083, 2018. Collection of open source public transport networks by DFG research unit FOR 2083: integrated planning for public transportation. https://github.com/FOR2083/PublicTransportNetworks.

Friedrich, M., Müller-Hannemann, M., Rückert, R., Schiewe, A., Schöbel, A., 2017. Robustness tests for public transport planning. In: D'Angelo, Gianlorenzo, Dollevoet, Twan (Eds.), 17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017). In: volume 59 of OpenAccess Series in Informatics (OASIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 6:1–6:16. http://dx.doi.org/10.4230/OASIcs.ATMOS.2017.6.

Friedrich, M., Müller-Hannemann, M., Rückert, R., Schiewe, A., Schöbel, A., 2018. Robustness as a third dimension for evaluating public transport plans. In: Borndörfer, R., Storandt, S. (Eds.), 18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018). In: volume 65 of OpenAccess Series in Informatics (OASIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 4:1–4:17. http://dx.doi.org/10.4230/OASIcs.ATMOS.2018.4.

Goerigk, M., Heße, S., Müller-Hannemann, M., Schmidt, M., Schöbel, A., 2013a. Recoverable robust timetable information. In: Frigioni, D., Stiller, S. (Eds.), 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems. In: volume 33 of OpenAccess Series in Informatics (OASIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 1–14. http://dx.doi.org/10.4230/OASIcs.ATMOS.2013.1.

Goerigk, M., Schachtebeck, M., Schöbel, A., 2013b. Evaluating line concepts using travel times and robustness: Simulations with the lintim toolbox. Public Transp. 5 (3), 267–284.

Goerigk, M., Schmidt, M., Schöbel, A., Knoth, M., Müller-Hannemann, M., 2014. The price of strict and light robustness in timetable information. Transp. Sci. 48, 225–242.

Goerigk, M., Schöbel, A., 2010. An empirical analysis of robustness concepts for timetabling. In: Erlebach, T., Lübbecke, M. (Eds.), 10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10). In: volume 14 of OpenAccess Series in Informatics (OASIcs), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, pp. 100–113.

Goerigk, M., Schöbel, A., 2014. Recovery-to-optimality: A new two-stage approach to robustness with an application to aperiodic timetabling. Comput. Oper. Res. 52, 1–15.

Goerigk, M., Schöbel, A., 2016. Algorithm engineering in robust optimization. In: Kliemann, L., Sanders, P. (Eds.), Algorithm Engineering: Selected Results and Surveys. In: volume 9220 of LNCS State of the Art, Springer, pp. 245–279.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press, http://www.deeplearningbook.org.

Google Brain Team, 2015. Tensorflow. https://www.tensorflow.org/.

Grafe, V., Schöbel, A., 2021. Solving the periodic scheduling problem: An assignment approach in non-periodic networks. In: M. Müller-Hannemann and F. Perea (Ed.), 21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2021), volume 96 of Open Access Series in Informatics (OASIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik. pp. 9:1–9:16, https://doi.org/10.4230/OASIcs.ATMOS.2021.9.

König, E., 2020. A review on railway delay management. Public Transp. 12 (2), 335–361.

Kuhn, M., Johnson, K., 2019. Feature Engineering and Selection — a Practical Approach for Predictive Models. CRC Press,Taylor & Francis Ltd.

Liebchen, C., Lübbecke, M., Möhring, R., Stiller, S., 2009. The concept of recoverable robustness, linear programming recovery, and railway applications. In: Robust and Online Large-Scale Optimization. In: volume 5868 of Lecture Notes in Computer Science, Springer, pp. 1–27.

Lindner, N., Liebchen, C., Masing, B., Forward cycle bases and periodic timetabling.In: M. Müller-Hannemann and F. Perea (Ed.), 21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2021), volume 96 of Open Access Series in Informatics (OASIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik., pp. 2:1–2:14, https://doi.org/10.4230/OASIcs.ATMOS.2021.2.

Lu, Q.-C., 2018. Modeling network resilience of rail transit under operational incidents. Transp. Res. Part A: Policy Pract. 117, 227–237. http://dx.doi.org/10.1016/j.tra.2018.08.015.

Lu, C., Tang, J., Zhou, L., Yue, Y., Huang, Z., 2017. Improving recovery-to-optimality robustness through efficiency-balanced design of timetable structure. Transp. Res. C 85, 184–210.

Lusby, R., Larsen, J., Bull, S., 2018. A survey on robustness in railway planning. European J. Oper. Res. 266 (1), 1–15.

Lusby, R., Larsen, J., Ehrgott, M., Ryan, D., 2011. Railway track allocation: models and methods. OR Spectrum 33 (4), 843–883.

Mahdavinejad, M.S., Rezvan, M., Barekatain, M., Adibi, P., Barnaghi, P., Sheth, A.P., 2018. Machine learning for internet of things data analysis: a survey. Digit. Commun. Netw. 4 (3), 161–175. http://dx.doi.org/10.1016/j.dcan.2017.10.002.

Matos, G., Albino, L., Saldanha, R., Morgado, E., 2021. Solving periodic timetabling problems with SAT and machine learning. Public Transp. 13, 625–648. http://dx.doi.org/10.1007/s12469-020-00244-y.

Müller-Hannemann, M., Rückert, R., Schiewe, A., Schöbel, A., 2021. Towards improved robustness of public transport by a machine-learned oracle. In: Müller-Hannemann, M., Perea, F. (Eds.), 21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2021, September (2021) 9-10, Lisbon, Portugal (Virtual Conference). In: volume 96 of OASIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 3:1–3:20. http://dx.doi.org/10.4230/OASIcs.ATMOS.2021.3.

Müller-Hannemann, M., Rückert, R., Schiewe, A., Schöbel, A., 2022. Framework for generating machine learning models for robustness. In: Homepage.https://gitlab.rlp.net/for2083/framework-for-generating-machine-learning-models-for-robustness.

Müller-Hannemann, M., Rückert, R., 2017. Dynamic event-activity networks in public transportation — timetable information and delay management. Datenbank-Spektrum 17, 131–137. http://dx.doi.org/10.1007/s13222-017-0252-y.

Nguyen, H., Kieu, L., Wen, T., Cai, C., 2018. Deep learning methods in transportation domain: a review. IET Intell. Transp. Syst. 12 (6), 998–1004, https://digital-library.theiet.org/content/journals/10.1049/iet-its.2018.0064.

Oliphant, T., 2016. Numpy - the fundamental package for scientific computing with python. https://numpy.org/.

Oneto, L., Fumeo, E., Clerico, G., Canepa, R., Papa, F., Dambra, C., Mazzino, N., Anguita, D., 2018. Train delay prediction systems: A big data analytics perspective. Big Data Res. 11, 54–64. http://dx.doi.org/10.1016/j.bdr.2017.05.002, Selected papers from the 2nd INNS Conference on Big Data: Big Data & Neural Networks.

Parbo, J., Nielsen, O., Prato, C., 2016. Passenger perspectives in railway timetabling: a literature review. Transp. Rev. 36 (4), 500–526.

Pätzold, J., 2021. Finding robust periodic timetables by integrating delay management. Public Transp. 13, 349–374. http://dx.doi.org/10.1007/s12469-020-00260-y.

Pätzold, J., Schiewe, A., Schiewe, P., Schöbel, A., 2017. Look-ahead approaches for integrated planning in public transportation. In: D'Angelo, Gianlorenzo, Dollevoet, Twan (Eds.), 17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017). In: volume 59 of OpenAccess Series in Informatics (OASIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik., , Dagstuhl, Germany, pp. 17:1–17:16. http://dx.doi.org/10.4230/OASIcs.ATMOS.2017.17.

Pätzold, J., Schöbel, A., 2016. A matching approach for periodic timetabling. In: Goerigk, Marc, Werneck, Renato (Eds.), 16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016). In: volume 54 of OpenAccess Series in Informatics (OASIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik., Dagstuhl, Germany, pp. 1:1–1:15. http://dx.doi.org/10.4230/OASIcs.ATMOS.2016.1.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830.

Peeters, L., Kroon, L., 2001. A cycle based optimization model for the cyclic railway timetabling problem. In: Computer-Aided Scheduling of Public Transport. Springer, pp. 275–296.

Polinder, G.-J., Breugem, T., Dollevoet, T., Maróti, G., 2019. An adjustable robust optimization approach for periodic timetabling. Transp. Res. B 128, 50–68.

Polinder, G.-J., Cacchiani, V., Schmidt, M., Huisman, D., 2020. An Iterative Heuristic for Passenger-Centric Train Timetabling with Integrated Adaption Times. Technical report, http://dx.doi.org/10.2139/ssrn.3629576.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V., Gulin, A., 2019. Catboost: unbiased boosting with categorical features. arXiv:1706.09516.

Raschka, S., 2015. Python Machine Learning. Packt Publishing, Birmingham, UK.

Rückert, R., Lemnian, M., Blendinger, C., Rechner, S., Müller-Hannemann, M., 2017. PANDA: a software tool for improved train dispatching with focus on passenger flows. Public Transp. 9 (1), 307–324. http://dx.doi.org/10.1007/s12469-016-0140-0.

Schiewe, A., Albert, S., Grafe, V., Schiewe, P., Schöbel, A., Spühler, F., LinTim: An integrated environment for mathematical public transport optimization.In: Documentation for Version 2021.12. Technical report.Fraunhofer-Institut für Techno- und Wirtschaftsmathematik, https://nbn-resolving.org/urn:nbn:de:hbz:386-kluedo-66870.

Schiewe, A., Albert, S., Grafe, V., Schiewe, P., Schöbel, A., Spühler, F., 2022. LinTim - integrated optimization in public transportation. In: Homepage. https://lintim.net.

Schmidhuber, J., 2015. Deep learning in neural networks: An overview. Neural Netw. 61, 85–117.

Schöbel, A., 2012. Line planning in public transportation: models and methods. OR Spectrum 34 (3), 491–510.

Schöbel, A., Kratz, A., 2009. A bicriteria approach for robust timetabling. In: Ahuja, R.K., Möhring, R.H., Zaroliagis, C.D. (Eds.), Robust and Online Large-Scale Optimization. In: volume 5868 of Lecture Notes in Computer Science, Springer, pp. 119–144.

Schöbel, A., Scholl, S., 2006. Line planning with minimal transfers. In: 5th Workshop on Algorithmic Methods and Models for Optimization of Railways, number 06901 in Dagstuhl Seminar Proceedings.

Schöbel, A., Schwarze, S., 2013. Finding delay-resistant line concepts using a game-theoretic approach. Netnomics 14 (3), 95–117. http://dx.doi.org/10.1007/s11066-013-9080-x.

Serafini, P., Ukovich, W., 1989. A mathematical model for periodic scheduling problems. SIAM J. Discrete Math. 2, 550–581.

Varghese, V., Chikaraishi, M., Urata, J., 2020. Deep learning in transport studies: A meta-analysis on the prediction accuracy. J. Big Data Anal. Transp. 2, 199–220. http://dx.doi.org/10.1007/s42421-020-00030-z.

Wang, Y., Zhang, D., Liu, Y., Dai, B., Lee, L.H., 2019. Enhancing transportation systems via deep learning: A survey. Transp. Res. C 99, 144–163. http://dx.doi.org/10.1016/j.trc.2018.12.004.

Wüst, R., Bütikofer, S., Ess, S., Gomez, C., Steiner, A., Laumanns, M., Szabo, J., 2019. Improvement of maintenance timetable stability based on iteratively assigning event flexibility in fpesp. In: 8th International Conference on Railway Operations Modelling and Analysis (ICROMA), NorrkÖPing, Sweden, June 17th–20th, 2019. In: RailNorrköping 2019, (069), Linköping University Electronic Press, pp. 1140–1157.

Yap, M., Cats, O., 2020. Predicting disruptions and their passenger delay impacts for public transport stops. Transportation 1–29. http://dx.doi.org/10.1007/s11116-020-10109-9.

Yap, M.D., van Oort, N., van Nes, R., van Arem, B., 2018. Identification and quantification of link vulnerability in multi-level public transport networks: a passenger perspective. Transportation 45 (4), 1161–1180.