# Towards Improved Robustness of Public Transport by a Machine-Learned Oracle

**Matthias Müller-Hannemann**
Martin-Luther-Universität Halle-Wittenberg, Germany

**Ralf Rückert**
Martin-Luther-Universität Halle-Wittenberg, Germany

**Alexander Schiewe**
TU Kaiserslautern, Germany

**Anita Schöbel**
Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM, Kaiserslautern, Germany
TU Kaiserslautern, Germany

## Abstract

The design and optimization of public transport systems is a highly complex and challenging process. Here, we focus on the trade-off between two criteria which shall make the transport system attractive for passengers: their travel time and the robustness of the system. The latter is time-consuming to evaluate. A passenger-based evaluation of robustness requires a performance simulation with respect to a large number of possible delay scenarios, making this step computationally very expensive.

For optimizing the robustness, we hence apply a machine-learned oracle from previous work which approximates the robustness of a public transport system. We apply this oracle to bi-criteria optimization of integrated public transport planning (timetabling and vehicle scheduling) in two ways: First, we explore a local search based framework studying several variants of neighborhoods. Second, we evaluate a genetic algorithm. Computational experiments with artificial and close to real-word benchmark datasets yield promising results. In all cases, an existing pool of solutions (i.e., public transport plans) can be significantly improved by finding a number of new non-dominated solutions, providing better and different trade-offs between robustness and travel time.

**2012 ACM Subject Classification** Applied computing → Transportation

**Keywords and phrases** Public Transportation, Timetabling, Machine Learning, Robustness

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2021.3

**Funding** This work has been partially supported by DFG under grants SCHO 1140/8-2 and MU 1482/7-2.

## 1 Introduction

The design and planning of public transport systems is a challenging, multi-faceted optimization problem. Given an infrastructure network (stations and direct connections), a line concept (a set of lines with corresponding frequencies), and a passenger demand (origin-destination pairs between which passengers wish to travel), we here focus on the integrated optimization of a timetable with a corresponding vehicle schedule. The resulting public transport plan shall be cost-efficient, attractive to passengers, and robust against different types of disturbances. As usual in multi-criteria optimization, we are especially interested in finding non-dominated solutions, i.e. solutions for which no other solution exists which is at least as good in all criteria and strictly better in at least one.

In recent years, many different robustness concepts have been proposed, for recent surveys see [12, 16]. All these concepts compute the robustness of public transport systems differently, but as was, e.g., stressed in [21], considering the passengers when evaluating the robustness is very important. To do that, there are several approaches, where especially the effect of disruptions on passengers are examined, see e.g., [4, 5, 8, 9, 15]. But such an assessment requires extensive performance simulations with respect to a large number of scenarios, each measuring the effect of specific delays on the passengers. Already a single robustness simulation is computationally very expensive, and therefore such an approach is hard to use within an iterative optimization framework. This motivates the design of a much faster, scenario-based robustness approximation by using methods from machine learning. In a recent paper, we developed an efficient oracle for the estimation of the robustness of a public transport plan by training an artificial neural network [19]. Based on only a few key features of a public transport plan, the trained neural network can be used as a black box to instantly predict the robustness of a previously unseen public transport plan with high accuracy.

**Contribution.** The long-term goal of this work is to provide an improved methodology for the planning of robust, but still efficient and attractive public transport systems. Here, we develop two algorithmic approaches, a local search and a genetic algorithm, both using the robustness oracle to iteratively compute competitive solutions w.r.t. passenger quality and robustness. First, we apply this oracle as a black box for increasing the robustness of a given timetable and a corresponding vehicle schedule by local search. We propose several alternative definitions of neighborhoods for this local search. Second, we develop a genetic approach that also uses the oracle to speed up the computations. Our experimental results are encouraging:

1. Local search, trying to improve robustness succeeds in most cases without worsening the average perceived travel time by too much.
2. More specifically, local search applied to a pool of non-dominated instances generates many new non-dominated solutions, thereby improving our approximation of the Pareto front significantly.
3. Similarly, our genetic approach shows clear improvements from a given starting population of solutions within few rounds.

**Related work.** Public transport planning consists of several stages that are traditionally solved sequentially. For this paper, we are considering *public transport systems*, namely a line plan, a timetable and a vehicle schedule. For an overview of line planning, see Schöbel [26], for an overview of timetabling, see Lusby et al. [17], and for an overview of vehicle scheduling, see Bunte and Kliewer [3].

There are several robustness concepts in literature. Due to the increase in complexity, methods for finding timetables based on these concepts often use heuristics, see e.g., Polinder et al. [23] or Pätzold [22] for recent approaches. For an overview of robust timetabling, see Lusby et al. [16]. Related to robust timetable creation is delay management, where trains are rescheduled in specific delay scenarios. For an overview of delay management, see Dollevoet et al. [6] and König [14]. Here, we consider the delay management strategy to be fixed and implicitly learnt by the robustness oracle. Hence, our used robustness evaluations are applicable to any given delay management strategy.

Both local search and genetic algorithms are used extensively in public transport research. See e.g., [10, 11, 13, 28] for local search applications and [1, 20, 27] for usages of a genetic algorithm approach. To our knowledge this paper is the first to use both approaches in the context of passenger flow-based robustness of timetables. There are other approaches of using machine learning (ML) in the optimization of public transport systems. For example, Matos et al. [18] use reinforcement learning for the optimization of periodic timetables and

Bauer and Schöbel [2] develop an approach to learn the quality of a connection in delay management. As far as we know, ML has not been used for robustness optimization in public transport before.

**Overview.** The remainder of this paper is structured as follows. In Section 2, we provide background information and basic notions used in this paper about public transport systems, sketch the evaluation of robustness, and briefly describe the machine learning approach by which we create an oracle for fast robustness evaluation. Then, Section 3 describes our local search framework and the different sets of neighborhoods used for optimization. In Section 4, we introduce a second approach based on a genetic algorithm. Experimental results are presented and discussed in Section 5. Finally, we conclude with an outlook.

## 2 Background: Public Transport, Robustness and Machine Learning

### 2.1 Public Transport Systems

To present our algorithmic approaches, we first need to clearly define the structures we are working with. For all our algorithms mentioned below, we will assume that an infrastructure network, a line concept and a passenger demand are given and cannot be changed. Here, a line concept is a set of paths through the infrastructure network, each with a frequency, i.e., a number of times the line should be served per planning period. We will call the infrastructure network with a given line concept and passenger demand a *dataset*. Overall, we want to determine the robustness of *instances*, i.e., a dataset combined with a timetable, a vehicle schedule and a corresponding set of passenger routes. As a basic underlying model, we use an event-activity network $(\mathcal{E}, \mathcal{A})$ with events $\mathcal{E}$, representing the departures and arrivals of vehicles at stops, and activities $\mathcal{A}$ between them. For activities, we are considering *drive*, *wait* and *turnaround* activities to model the vehicle behavior and *change* activities for transferring of passengers. Additionally, we assume that for every activity $a \in \mathcal{A}$ a lower bound $l_a$ and an upper bound $u_a$ are given, determining the feasibility of the timetable. Depending on the context, different sets of change activities will be considered. For timetable construction, we consider as change activities a small set of important transfer possibilities which shall be guaranteed. Afterwards, i.e., for evaluating a timetable, we allow passengers to use all possible transfers as change activities. A feasible *timetable* now assigns a time $\pi_e$ to each event $e \in \mathcal{E}$ such that the duration $d_a$ of every activity $a \in \mathcal{A}$ stays within the given bounds. We are considering both periodic and aperiodic timetables, depending on the algorithm used.

A *trip* is a path of drive and wait activities in the event-activity-network that needs to be operated by a single vehicle. The *vehicle schedule* is a collection of vehicle tours, each covering a set of trips. To be feasible, each trip in the event-activity network needs to be covered exactly once and the corresponding turnaround activities are feasible w.r.t. their bounds, i.e., the last event of a trip and the first event of the consecutive trip by the same vehicle have enough time between them, e.g., to drive from one station to the other.

The last objects to consider are the *passenger routes*. We use the given passenger demand data, with corresponding earliest departure times for each passenger, to determine a realistic passenger routing. To achieve this, scarce vehicle capacities are also very important. Each passenger chooses a route that optimizes a utility function. For this, we use a model where the passengers are searching for their shortest paths w.r.t. the perceived travel time, i.e., a weighted sum of travel time and the number of transfers (e.g. by counting 5 minutes per transfer), while respecting the capacity of the vehicles. Conflicts are resolved using seat-reservation in a first come first serve order, i.e., once a passenger chooses her path the

**Table 1** Robustness tests RT-1-RT-4 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 minutes |
| RT-3 | temporary blocking of single station | emulates a gridlock at a station | blocking of 15 minutes |
| RT-4 | random delay simulation | emulates multiple common independent delays | empirical distribution of delays based on [9] |

corresponding capacity in the vehicle is guaranteed. In our experiments, this model is much faster than a more complex simulation involving capacity checks when boarding a vehicle but provides nearly the same results. See [19] for a more detailed discussion of these models.

## 2.2 Robustness

The evaluation of the robustness of public transport networks we use is based on Friedrich et al. [8, 9]. With a simulation framework, we conduct several robustness tests simulating certain aspects of common disturbances during daily operation. During this process, we measure the arrival times of all passengers compared to their initially planned arrival time. The sum of these differences for all passengers provides the *robustness value* of the simulation. This value serves as an orientation for comparing public transport plans using the same passenger demand. For better interpretability and comparability, however, we normalize the robustness values for a set of known instances to a scale from 0 to 100 where 100 is the worst instance. Hence, smaller robustness values are better.

In [9], we give a very detailed description of all aspects of the four tests we use here. To better illustrate this method we now give a detailed explanation of the first robustness test RT-1. The task of RT-1 is to simulate the total effect that starting delays have on the schedule. The delay caused when the first departure of a vehicle is not on time is a common occurrence in daily operation. To evaluate this metric, RT-1 creates a separate scenario where each vehicle has a delay of $x$ minutes and all other vehicles are on time. The sum of all passenger delays at their destination is the final result of RT-1. The result of this test is deterministic but highly dependent on the parameters specifying the passenger and the delay management models. So if, for example, each vehicle waits for transferring passengers or passengers neglect maximum vehicle capacities this produces another specific robustness value. In Table 1, we provide a brief description of the considered robustness tests.

## 2.3 Robustness Estimation by Machine Learning

Conducting the four robustness tests mentioned in the last section is computationally expensive. In an optimization scenario where parts of an instance are altered we want to know the effect on the robustness value as quickly as possible. To this end, we want to approximate the real robustness by using an oracle as a predictor based on machine learning. In [19] we first introduced such an oracle and evaluated its approximation performance. In this section, we will briefly explain how this oracle is created and how we use it for robustness approximation. First, we give a short overview of how the process works, explaining the most important steps. The creation of the oracle can be done in four steps:

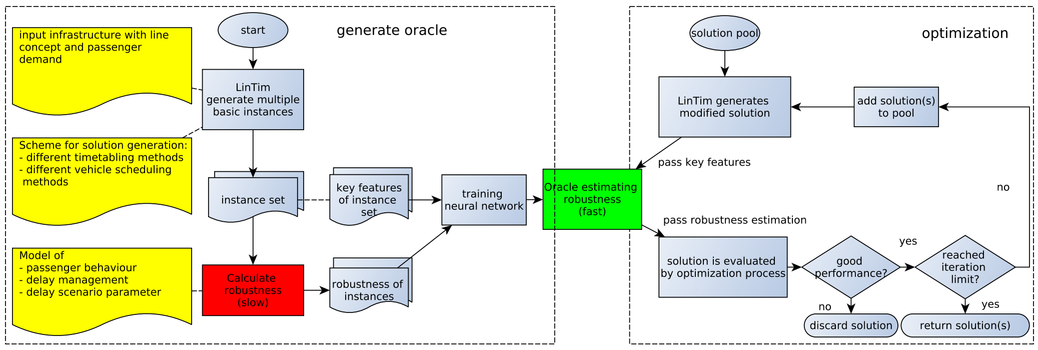

generate oracle
start
input infrastructure with line concept and passenger demand
LinTim generate multiple basic instances
Scheme for solution generation:
- different timetabling methods
- different vehicle scheduling methods
instance set
key features of instance set
training neural network
Model of
- passenger behaviour
- delay management
- delay scenario parameter
Calculate robustness (slow)
robustness of instances
Oracle estimating robustness (fast)
optimization
solution pool
LinTim generates modified solution
add solution(s) to pool
pass key features
no
pass robustness estimation
solution is evaluated by optimization process
good performance?
yes
reached iteration limit?
no
yes
discard solution
return solution(s)

**Figure 1** Left box: Workflow of the creation of the oracle for estimating robustness of a public transport system by training a neural network. Right box: Optimization as an exemplary application of the oracle. Yellow fields denote input or choices of models and methods which are specific for each application (but can easily be adapted).

**Table 2** Our selection for key features and their length, which is specific for each infrastructure network. In our networks the maximal values were 240 minutes for the maximal travel time $\text{traveltime}^{\max}$, 10 for the maximal number of transfers $\#\text{transfers}^{\max}$ and 30 minutes for the maximal turnaround time $\text{turnaround}^{\max}$. $m$ is the number of infrastructure edges, $n$ the number of stations.

| # | 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 per station | $n$ |
| 7 | the average sum of line frequencies per station | $n$ |
| 8 | the share of events per station | $n$ |
| 9 | the number of trips with an outgoing turnaround slack of $i$ minutes | $\text{turnaround}^{\max}$ |

1. defining a set of key features representing an instance,
2. generating a large number of training instances,
3. calculating the robustness of the instances with the original robustness test,
4. using ML to estimate robustness only by knowing the key features.

Figure 1, adapted from [19], illustrates how the creation and usage of the oracle is linked to the optimization process. The definition of a set of key features is essential for several reasons. We want a compact way of representing an instance with a fixed number of elements so ML-algorithms can easily use this as input. If this is achieved the creation of the key features during the optimization can be done fast without transferring large data sets to the oracle. Characteristic features include slack values on activities, occupancy rates on vehicles, and the number of passengers using a transfer. For a detailed list of the key features we selected for our model see Table 2.

In the second step, we need to create a large number of instances as the training set for the ML-algorithm. To do so, we use several different timetabling and vehicle scheduling methods as well as buffering strategies, provided by the open-source library LinTim [24, 25]. These instances should ideally be diverse and cover most robustness values. If there are gaps where we have no instances associated with a certain interval of robustness values (as can

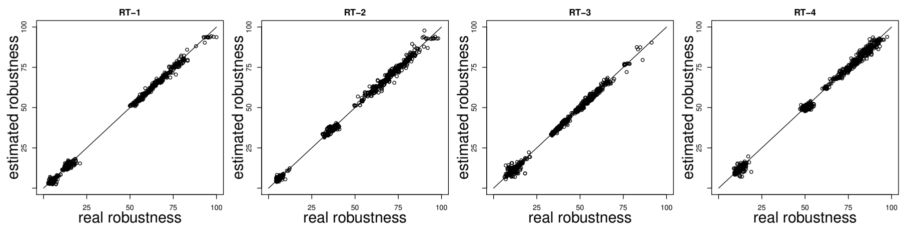


**Figure 2** Predictions of all four robustness tests for all instances of the `grid` network [19].

be observed in Figure 2), new instances may not accurately be predicted near this range, see [19] for details. In the third step, the robustness values for the training set need to be calculated. This is done by the framework mentioned in Section 2.2. In the last step, the training of the oracle is done. We selected an artificial neural network (ANN) for the task of machine learning. More specifically, we use a neural network with five hidden layers and an output layer with four neurons predicting each of the four robustness tests separately. The network was trained using a training-, test- and validation set. For all our four infrastructure networks we achieved an average error below 1% in terms of the combined robustness value of all four robustness tests [19] (see Figure 2). However, the oracle performed worse when tested with instances created with a different method and robustness values outside the clusters in the training set. In spite of these limitations, we will see that the oracle is powerful enough to guide local search and the genetic algorithm into the desired direction.

## 3 Improving Robustness by Local Search

Our first approach in utilizing the robustness oracle presented in Section 2.3 is a generic local search approach. It was first described in [19] and is stated in Algorithm 2, Appendix A. The main idea is to determine a local neighborhood of the current solution in each step, evaluating all solutions in the neighborhood using the robustness oracle.

The algorithm has a given instance as a starting solution, i.e., a fixed dataset, consisting of an infrastructure network, a line concept and a passengers' demand, in combination with a timetable, a vehicle schedule and corresponding passenger routes. Since the dataset is fixed, we want to improve the robustness of the starting solution by changing the timetable and the vehicle schedule. To do so, we compute a local neighborhood of timetables in each step, introducing possible changes to the timetable. Note that this may include changes of the duration of turnaround activities, therefore requiring to adapt the vehicle schedule as well.

For every neighborhood, we consider several different activities for which an increase of its slack (i.e. the difference of planned duration and lower bound) could benefit the robustness of the instance: We sort the wait, drive and change activities each based on their current slack, divided by the number of passengers using the activity in the current passenger routes, and the turnaround activities by their current slack. Obtaining the $N$ (here: $N = 20$) activities with the smallest weight from every sorted list, we get a candidate set of $4N$ activities. For each candidate $a = (i, j) \in \mathcal{A}$, $i, j \in \mathcal{E}$ with a lower bound $l_a$ and a current duration $d_a$, we then increase the slack $d_a - l_a$ of the activity, resulting in a later time for the target event $j$.

This resulting timetable may be infeasible, since the lower bound on some activities $(j, k) \in \mathcal{A}$, $k \in \mathcal{E}$ may not be respected anymore. Therefore, we need a propagation strategy to reconstruct a feasible timetable. We considered the following four strategies here:

**Strategy 1:** Use all original slack. For every infeasible activity $a = (i, j) \in \mathcal{A}$, we increase the target event time exactly as much such that the lower bound of the activity is fulfilled again, i.e., $\pi_j = \pi_i + l_a$. Thus, the slack of this activity is reduced to zero.

**Strategy 2:** Reuse no slack. We maintain the original slack on each activity $a = (i, j) \in \mathcal{A}$, therefore shifting the complete timetable after the considered candidate, i.e., $\pi_j = \pi_i + d_a$.

**Strategy 3:** Reuse $J$% (here: $J = 50$) of the original slack, shifting the target event time of an infeasible activity $a = (i, j) \in \mathcal{A}$ according to $\pi_j = \pi_i + l_a + \frac{J}{100} \cdot (d_a - l_a)$.

**Strategy 4:** Reuse some slack, maintaining a minimal slack of $K$ seconds (here: $K = 300$) (or the original slack, if it was less) on $a = (i, j) \in \mathcal{A}$, i.e., $\pi_j = \pi_i + l_a + \min(K, d_a - l_a)$

Note that we are considering all possibly infeasible activities here, including all change activities that are used by some passenger. The latter implies that passenger routes remain feasible. For ease of presentation, we do not check the upper bounds of the activities, assuming that we can postpone all events arbitrarily. If the upper bounds $u_a$ should be considered as well, the above equations can be easily adapted to include the corresponding constraints. It may be the case that this leads to infeasibilities that can not be handled by our propagations strategies. In these cases we simply remove the candidate from the neighborhood. The different strategies lead to different trade-offs between robustness and travel time of the passengers, as can be seen in the computational experiments in Section 5.

Another aspect of the propagation strategy is whether to consider aperiodic or periodic timetables: The formulas given above can be extended to the periodic case, allowing us to maintain a feasible periodic timetable where we need to shift all corresponding periodic events at once. This leads to much larger changes in the resulting timetable and an additional travel time increase for the passengers. Furthermore, the reconstruction of feasible solution may not be possible due to the additional periodicity constraint. In such cases, we again remove the candidate from the neighborhood. Mathematically, the case where we allow an aperiodic timetable is a relaxation of the periodic case, therefore allowing better solutions w.r.t. robustness. We provide computational evaluations for both cases in Section 5.

After every candidate has a restored feasible timetable, we can use the oracle to predict the robustness value of the corresponding instance. Additionally, we evaluate the current passenger routes, rejecting candidates where the travel time of the passengers increases too much. The resulting set of instances serves as the neighborhood set for the local search and we can choose the best solution in terms of estimated robustness as the new current solution.

To improve the runtime of the local search, we do not update the passenger routes in every step. Since we maintain a feasible timetable, all passenger routes remain feasible as well but may be suboptimal for single passengers. The idea is that for every single iteration, the changes in the corresponding timetable are not too big, therefore not changing the optimal passenger routes too drastically. To maintain an accurate robustness prediction, we introduce additional rerouting steps, where we recompute all passenger routes every few iterations and therefore improve the accuracy of the robustness oracle.

## 4 Genetic Algorithm

Our second approach applying the robustness oracle is a genetic algorithm. Feasible solutions, i.e., instances as defined above, are mutated and breed to create new and hopefully better solutions w.r.t. robustness. The general procedure is described in Algorithm 1.

To allow an easy mutation and breeding of different instances without losing feasibility, we choose a specific data model to represent solutions in a compact way as genes of equal length. Every current solution in our algorithm is determined by a vector $s$ of slacks for each possible activity in the event-activity-network, as well as a set of passenger routes. This

**Algorithm 1** Genetic algorithm using machine learning.

```
Data: the starting solution set currentSolutions
currentSolutions = currentSolutions ∪ mutate(currentSolutions)
while iteration limit not reached do
    if Rerouting step? then
        Reroute passengers in all solutions and update currentSolutions
    end
    currentSolutions = breed(currentSolutions)
    currentSolutions = predictAndSelect(currentSolutions)
end
Result: currentSolutions
```

allows for easy mutation and breeding, since every non-negative slack vector can be converted into a feasible timetable, by propagating the slack from a given start event. Note that we again do not consider upper bounds on the activities here and that we deliberately omit the vehicle schedule. Since optimal vehicle schedules can be computed very fast, these are calculated ad hoc when needed. To mutate an instance, a given number of $l$ (here: $l = 100$) entries in the vector $s$ are randomly selected and the corresponding slack value is changed by a random value in $[-m, m]$ for a given $m$ (here: $m = 120$ seconds), provided that the updated slack remains non-negative. For breeding, we choose two parents randomly from the previous generation and combine their slack vectors, i.e., for each entry in the slack vector we randomly decide which of the possible values to use. This is done $n$ times where $n$ is given beforehand, i.e., we gain $n$ new individuals for each generation. We choose a rather low number of 10 as the generation size and number of breedings per iteration due to the relatively large amount of memory needed to store the different entities. Additionally, the child is mutated as described above and directly inherits the passenger routes of one parent.

After creating the next generation in the breeding step, we introduce a selection process, reducing the number of candidates to a given $g$. In our implementation, both parents and children enter the selection process. We tested different variants here: The *quality* strategy selects solutions solely based on their estimated robustness, ignoring the travel time for the passenger. The *Pareto* strategy on the other hand chooses the non-dominant solutions in the current solution pool, i.e., solutions with a worse estimated robustness may remain in the population if their travel time is good enough. If less than $g$ solutions are non-dominated, we choose the best non-selected solutions w.r.t. the robustness estimation to fill up the next generation. The difference between the two strategies is discussed in Section 5.

Note that since we only store the timetable for each solution, we need to compute a new vehicle schedule for each evaluation. To do so, a flow-based integer programming formulation of the open-source software library LinTim [24, 25] was used. Additionally, to maintain realistic passenger routes, we add a rerouting step that is executed every few iterations, computing new optimal passenger routes for every instance in the current generation.

## 5 Experiments

Algorithms 1 and 2 and their beforehand discussed variants were implemented and tested on several datasets: two artificial benchmark datasets, `grid` and `ring`, see [7], and two close-to real world datasets, the bus system in Göttingen, Germany (`goevb`) and the regional train network in southern Lower Saxony, Germany (`lowersaxony`). All datasets are available as part of the open-source library LinTim, see [24, 25]. Their key features are given in Table 3, for a visualization of the infrastructure networks see Appendix B.

**Table 3** 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 |

**Table 4** Average improvement of the different propagation strategies using 120 seconds of slack on all datasets and starting instances.

| strategy | avg. robustness change | avg. perceived travel time change |
|---|---|---|
| Strategy 1 | -7.00% | +0.27% |
| Strategy 2 | -0.97% | +2.17% |
| Strategy 3 | -10.66% | +0.28% |
| Strategy 4 | -5.44% | +1.25% |

## 5.1 Local Search

First, we discuss our evaluations of the local search algorithm, presented in Section 3. We study the different neighborhoods resulting from the propagation strategies, as well as the potential to improve the pool of existing solutions. The following experiments were run for several different starting instances per dataset. For the presentation, we selected one instance with small, medium and high initial robustness values, respectively.

### Propagation Strategy

To evaluate the local search, we first discuss the different propagation strategies proposed in Section 3. In Table 4, the average changes to the two objective functions are given for all considered starting instances on all datasets. On average, Strategy 3 provides the best trade-off, by significantly improving the robustness at a small expense of increasing travel time. Figure 3 shows all different strategies used on three starting instances for `grid`. We can see that the expected performance, i.e., an improvement in robustness and an increase in passenger travel time can be observed for all cases. Furthermore, we see that the different strategies provide different trade-offs, e.g., Strategy 3 being a non-dominated solution (w.r.t. the other propagation strategies) for the starting solution with high robustness but not for the starting solution with the middle robustness. Note that although the improvement in robustness does not look significant for the starting solution with good robustness, the relative robustness improvement is still high, e.g., 31% for Strategy 4 compared to around 50% for Strategy 2 on the other two starting instances. Additionally, we see that choosing a smaller slack increase does not significantly alter the results obtained by the local search.

### Using non-dominated start instances

Next, we want to consider the overall quality of the solutions found by the local search. Since the quality of the solution is dependent on the starting instance, Figure 4 shows the effect of using the local search on every non-dominated original instance, i.e., on every original instance that is not dominated by another one. For this, we chose an initial slack increase of 120s per iteration and propagation Strategy 4. We can see that we find a huge

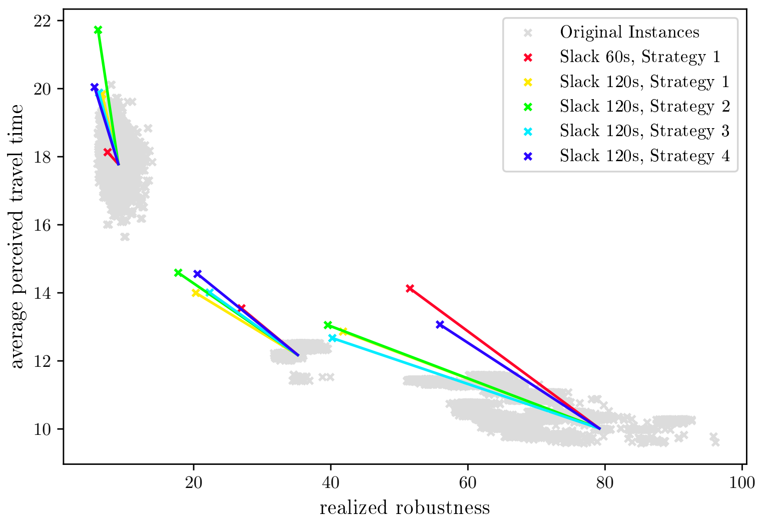
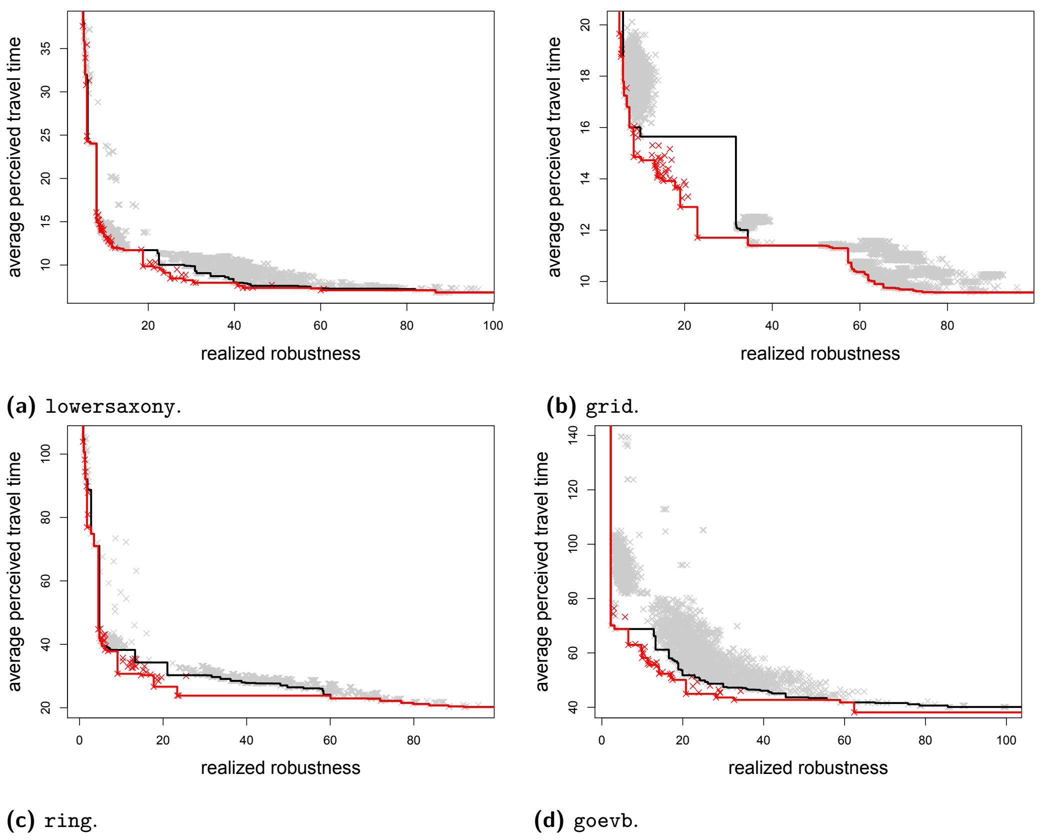


**Figure 3** Different propagation strategies for the local search, evaluated on `grid`. The performance is depicted by a line from the starting instance to the end result, where the end result is additionally marked by an "x". The strategies are given by their number from Section 3 and the slack increase in each iteration.



**(a)** `lowersaxony`.

**(b)** `grid`.

**(c)** `ring`.

**(d)** `goevb`.

**Figure 4** Aperiodic case: Using the local search on all non-dominated original instances. Old instances are grey, local search solutions are marked in red.

**Table 5** Aperiodic case: Sizes of the approximated Pareto sets using local search.

| dataset | originally non-dominated | together non-dominated | of those # new |
|---|---|---|---|
| `grid` | 51 | 58 | 14 |
| `ring` | 44 | 29 | 12 |
| `lowersaxony` | 50 | 50 | 21 |
| `goevb` | 38 | 18 | 14 |

**Table 6** Periodic case: Average improvement of the different periodic propagation strategies using 120 seconds of slack on all datasets and starting instance.

| strategy | avg. robustness change | avg. perceived travel time change |
|---|---|---|
| Strategy 1 | -2.87% | +0.14% |
| Strategy 2 | -0.09% | +0.01% |
| Strategy 3 | -2.2% | +0.07% |
| Strategy 4 | -0.7% | +0.04% |

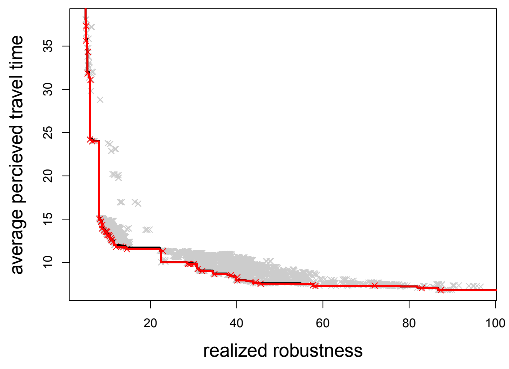
amount of solutions with structures not used beforehand, i.e., that have objective values that are very different from the original instances. This is especially true for `grid`, Figure 4b, where we have a large number of solutions in between the original clusters. But we find competitive results for all instances, now dominating multiple beforehand non-dominated solutions. An extreme example is `goevb`, where almost all originally non-dominated instances are dominated by local search solutions, namely 34 of 38 instances. An overview of the number of non-dominated solutions in the different solution sets can be found in Table 5.

## Periodic Timetabling and Local Search

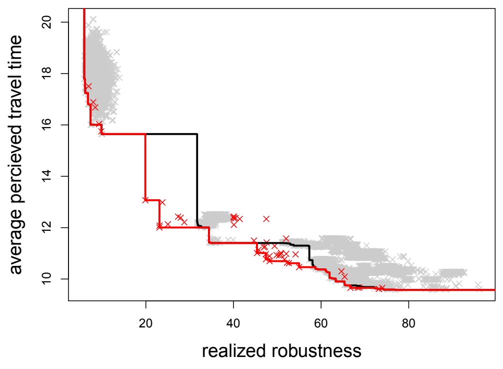
If we restrict the local search to finding periodic timetables, the algorithm can still improve the robustness of the start instances. In Table 6, the average changes to the two objective functions are given for all considered starting instances on all datasets. The number of new solutions found that are non-dominated can be seen in Figure 5. Table 7 is a visualization of the new Pareto fronts. Contrary to the aperiodic case, compare Table 4, there is no dominant solution on average, i.e., on average all strategies provide different trade-offs. But the amount of change in the two objective functions is smaller when compared to the aperiodic case, due to the additional periodic restrictions. As can e.g. be seen in Figure 6, using the periodic local search on a starting instance with middle robustness improves the only robustness by 40% instead of the 50% of the aperiodic case. Still, the periodic local search is able to improve the robustness of every given starting instance on dataset `grid`.

**Table 7** Periodic case: Sizes of the approximated Pareto sets using periodic local search with 120s slack on the different datasets.

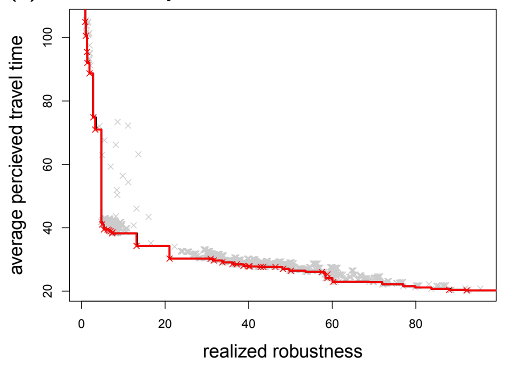
| dataset | originally non-dominated | together non-dominated | of those # new |
|---|---|---|---|
| `grid` | 51 | 40 | 12 |
| `ring` | 44 | 47 | 14 |
| `lowersaxony` | 50 | 63 | 28 |
| `goevb` | 38 | 39 | 3 |



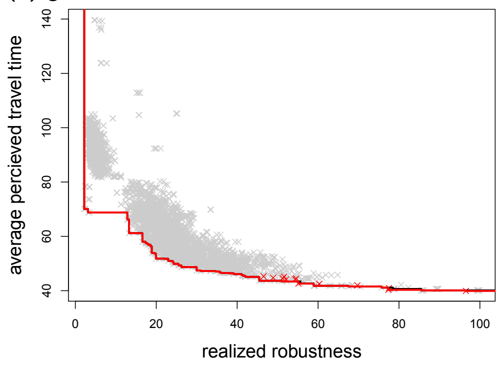
**(a)** `lowersaxony`.



**(b)** `grid`.



**(c)** `ring`.



**(d)** `goevb`.

**Figure 5** Periodic case: Approximated Pareto fronts for the solutions computed by the periodic local search with 120s slack on the different datasets. Old instances are grey, new solutions are marked in red.

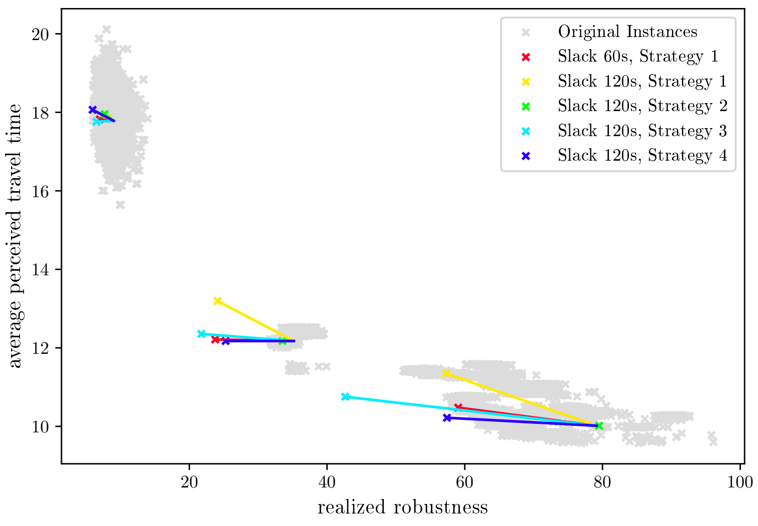


**Figure 6** Using periodic planning for the local search, evaluated on `grid`. The performance is depicted by a line from the starting instance to the end result, where the end result is additionally marked by an "x". The strategies are given by their number from Section 3 and the slack increase in each iteration.

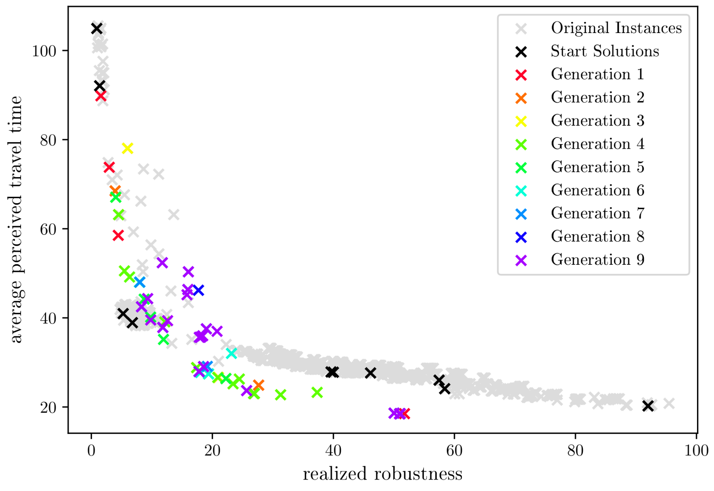


**Figure 7** Using the Pareto selection for the genetic algorithm, evaluated on `ring`.

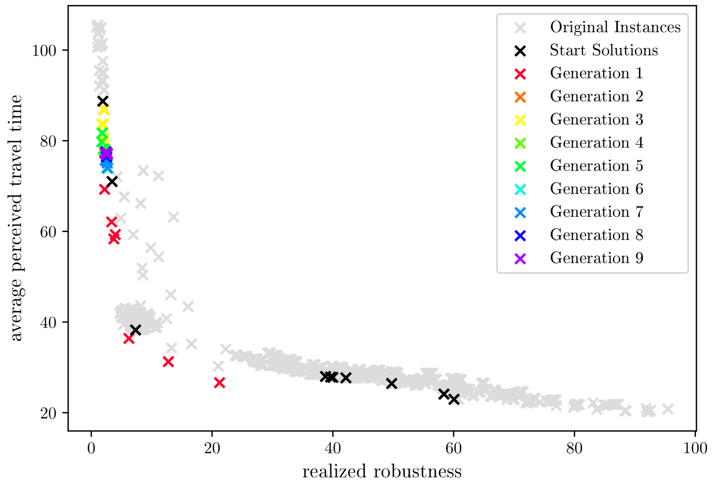


**Figure 8** Using the quality selection for the genetic algorithm, evaluated on `ring`.

## 5.2 Genetic Algorithm

The genetic algorithm was evaluated on the described datasets as well. Note that all experiments discussed here were repeated multiple times, due to their randomness. But since the behavior of different runs were similar, only one run is presented for each experiment.

### Choice of the selection process

The choice of the selection process shows the different qualities of the genetic algorithm. When using the Pareto selection, shown in Figure 7, the genetic algorithm produces multiple solutions dominating the original instances, exploring a large area of the previously empty part of the solution space. This produces several new competitive solutions for a decision maker to choose from. On the other hand, using the quality selection, shown in Figure 8, allows the genetic algorithm to focus on the estimated robustness value of the solution, producing more robust solutions with a higher travel time. Therefore both selection strategies have their advantages, the best strategy is dependent on the desired outcome of the algorithm.

**Table 8** Sizes of the approximated Pareto sets using the genetic algorithm.

| dataset | originally non-dominated | together non-dominated | of those # new |
|---|---|---|---|
| `grid` | 51 | 37 | 34 |
| `ring` | 44 | 48 | 43 |
| `lowersaxony` | 50 | 76 | 68 |
| `goevb` | 38 | 35 | 33 |

But we also see a disadvantage of the genetic algorithm: Since the algorithm only optimizes the estimated robustness value, it is dependent on the quality of the robustness oracle used. As was already discussed in [19], using the oracle in unexplored solution space potentially increases the error, complicating the computation of robust solutions. This can, e.g., be seen in the variance of the last generation in Figure 7. But nevertheless, the overall quality, i.e., the real simulated robustness, of the computed solutions is very high.

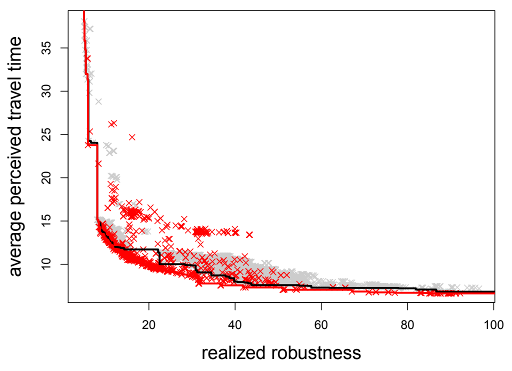
## Comparison to local search

To compare the genetic algorithm results with the local search results, we choose a depiction similar to Figure 4. In Figure 9, we collect the different solutions computed for the genetic algorithm experiments. With this, we can compare the approximated Pareto front of the different sets, namely the original instances, the local search solutions in Figure 4 and the genetic algorithm solutions. The genetic algorithm is able to compute a large set of competitive solutions, dominating even more original instances than the local search. For an overview of the number of non-dominated solutions, see Table 8. Especially in the area with worse robustness, the Pareto selection strategy combined with the randomness of the genetic algorithm results in a higher density of solutions. The genetic algorithm is therefore not only able to compute solutions with a good robustness but with very different trade-offs between robustness and passenger quality. Overall, both algorithms presented here are competitive and serve different means: While the local search can improve a single given starting solution w.r.t. the robustness value, the genetic algorithm is able to compute competitive solutions with different trade-offs from a set of given starting solutions.

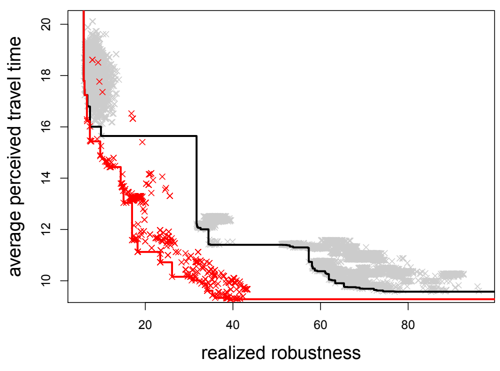
## Operating costs

Up until this point, we did not mention the operating cost of solutions since they are not in the focus of this work and we do not try to optimize them. But clearly robust and fast solutions still need to have competitive operating costs to be chosen by any public transport planner. Here, we only calculate and evaluate operational cost a posteriori.

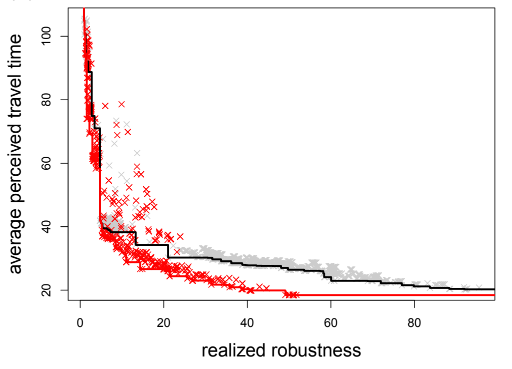
LinTim includes operating costs based on the number of vehicles used, driven kilometers and an additional cost per hour for every vehicle in use. In our experiments the corresponding parameters were set to 100000 € per vehicle, 1.5 € per kilometer and 25 € per hour. Figure 10 shows the Pareto fronts concerning cost and robustness for the aperiodic local search with 120s slack. The networks `grid` and `goevb` show several clusters of solutions where the costs are dominated by solutions inside clusters near the Pareto front. We can observe that several of the new solutions have costs that are competitive and belong to the Pareto-front.



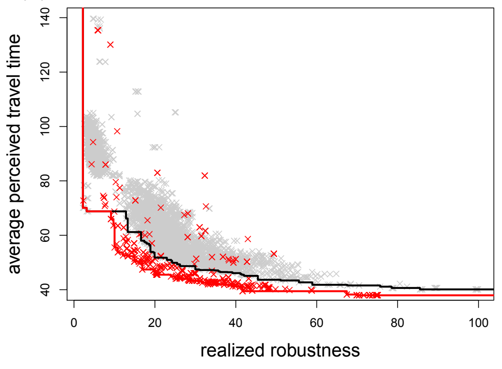
**(a)** `lowersaxony`.



**(b)** `grid`.



**(c)** `ring`.



**(d)** `goevb`.

**Figure 9** Approximated Pareto fronts for the solutions computed by the genetic search. Old instances are grey, new solutions are marked in red.

## 6 Outlook

In this paper, we have focused on improving the robustness of public transport systems from a passenger-oriented point of view. Our computational tests with local search and genetic algorithms demonstrated the ability of both methods to generate many new non-dominated solutions. However, there are still several improvements to consider. With respect to local search, we may further extend the definition of the used neighborhood and may consider combinations of several ones. For the genetic algorithm, next to changes in the selection process and the choice of starting instances, different mutation strategies would be possible to consider as well. This may further improve the exploration of the solution space, leading to more competitive solutions.

Improvement of the oracle and retraining is also of high importance. We need to eliminate gaps in the codomain, which are the robustness values. During the optimization using the genetic algorithm, we discovered many such solutions inside these gaps. This can be observed in Figure 9(b) where the space between 20 and 35 in the realized robustness objective is now populated. New solutions could now be added to the training process of the oracle, potentially allowing a better robustness estimation for future runs of the algorithms.

We plan to continue this line of work to see if similar results are possible when we modify the line concept, which is currently assumed to be fixed. Changing it would lead to different solution structures to learn for the oracle, extending the covered area in the solution space. Future work may also include further metaheuristics and stochastic local search methods.

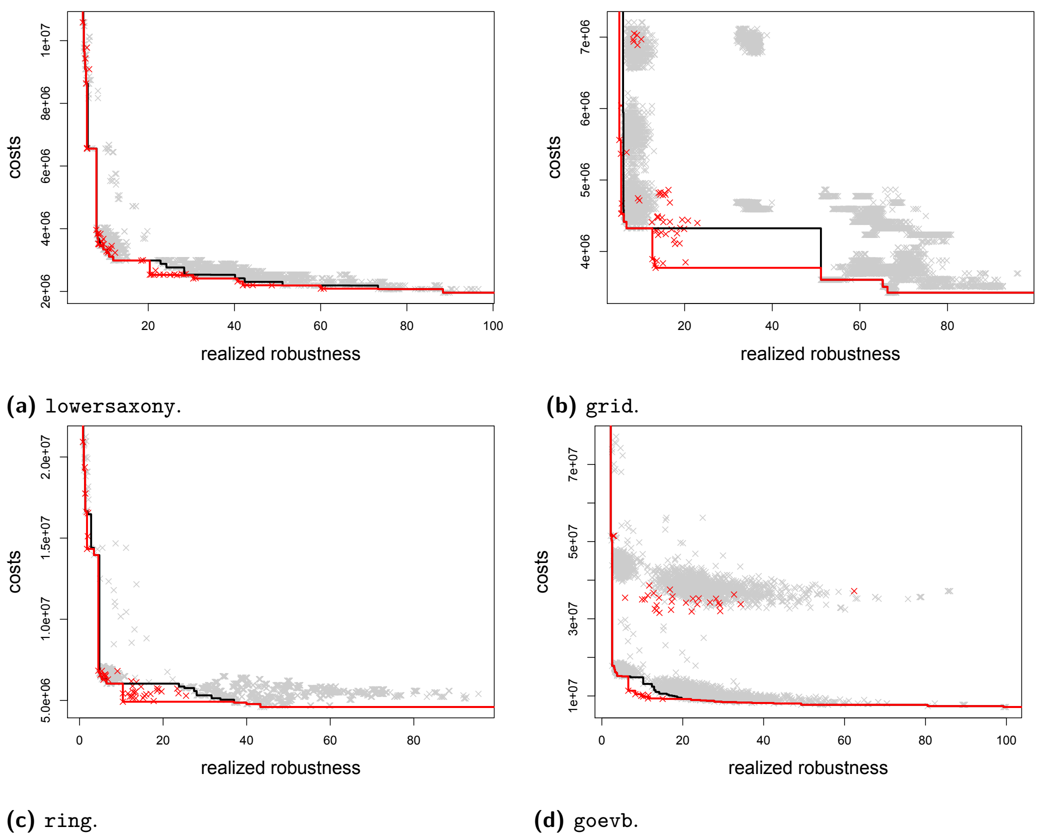


**Figure 10** Approximated Pareto fronts (robustness vs. operational costs) for the solutions computed by the aperiodic local search with 120s slack and strategy 4 on the different datasets. Old instances are grey, new solutions are marked in red.

## References

1 D. Arenas, R. Chevrier, S. Hanafi, and J. Rodriguez. Solving the train timetabling problem, a mathematical model and a genetic algorithm solution approach. In *6th international conference on railway operations modelling and analysis (RailTokyo2015)*, 2015.

2 R. Bauer and A. Schöbel. Rules of thumb — practical online strategies for delay management. *Public Transport*, 6(1):85–105, 2014.

3 S. Bunte and N. Kliewer. An overview on vehicle scheduling models. *Public Transport*, 1(4):299–317, 2009.

4 O. Cats. The robustness value of public transport development plans. *Journal of Transport Geography*, 51:236–246, 2016.

5 A. De-Los-Santos, G. Laporte, J. A. Mesa, and F. Perea. Evaluating passenger robustness in a rail transit network. *Transportation Research Part C: Emerging Technologies*, 20(1):34–46, 2012. Special issue on Optimization in Public Transport+ISTT2011. `doi:10.1016/j.trc.2010.09.002`.

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

7 Collection of open source public transport networks by DFG Research Unit “FOR 2083: Integrated Planning For Public Transportation”, 2018. URL: `https://github.com/FOR2083/PublicTransportNetworks`.

**8** M. Friedrich, M. Müller-Hannemann, R. Rückert, A. Schiewe, and A. Schöbel. Robustness Tests for Public Transport Planning. In G. D'Angelo and T. Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 6:1–6:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/OASIcs.ATMOS.2017.6`.

**9** M. Friedrich, M. Müller-Hannemann, R. Rückert, A. Schiewe, and A. Schöbel. Robustness as a Third Dimension for Evaluating Public Transport Plans. In R. Borndörfer and S. Storandt, editors, *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*, volume 65 of *OpenAccess Series in Informatics (OASIcs)*, pages 4:1–4:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/OASIcs.ATMOS.2018.4`.

**10** M. Goerigk. Exact and heuristic approaches to the robust periodic event scheduling problem. *Public Transport*, 7(1):101–119, 2015.

**11** M. Goerigk and A. Schöbel. Improving the modulo simplex algorithm for large-scale periodic timetabling. *Computers & Operations Research*, 40(5):1363–1370, 2013.

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

**13** O. Ibarra-Rojas, F. López-Irarragorri, and Y. Rios-Solis. Multiperiod bus timetabling. *Transportation Science*, 50(3):805–822, 2016.

**14** E. König. A review on railway delay management. *Public Transport*, 12(2):335–361, 2020.

**15** Q.-C. Lu. Modeling network resilience of rail transit under operational incidents. *Transportation Research Part A: Policy and Practice*, 117:227–237, 2018. `doi:10.1016/j.tra.2018.08.015`.

**16** R. Lusby, J. Larsen, and S. Bull. A survey on robustness in railway planning. *European Journal of Operational Research*, 266(1):1–15, 2018.

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

**18** G. Matos, L. Albino, R. Saldanha, and E. Morgado. Solving periodic timetabling problems with SAT and machine learning. *Public Transport*, 2020. `doi:10.1007/s12469-020-00244-y`.

**19** M. Müller-Hannemann, R. Rückert, A. Schiewe, and A. Schöbel. Estimating the robustness of public transport systems using machine learning, 2021. `arXiv:2106.08967`.

**20** K. Nachtigall and S. Voget. A genetic algorithm approach to periodic railway synchronization. *Computers & Operations Research*, 23(5):453–463, 1996.

**21** J. Parbo, O. Nielsen, and C. Prato. Passenger perspectives in railway timetabling: a literature review. *Transport Reviews*, 36(4):500–526, 2016.

**22** J. Pätzold. Finding robust periodic timetables by integrating delay management. *Public Transport*, 2021. `doi:10.1007/s12469-020-00260-y`.

**23** G. Polinder, V. Cacchiani, M. Schmidt, and D. Huisman. An iterative heuristic for passenger-centric train timetabling with integrated adaption times. ERIM Report Series Research in Management ERS-2020-006-LIS, Erasmus Research Institute of Management (ERIM), ERIM is the joint research institute of the Rotterdam School of Management, Erasmus University and the Erasmus School of Economics (ESE) at Erasmus University Rotterdam, 2020. URL: `https://ideas.repec.org/p/ems/eureri/127816.html`.

**24** A. Schiewe, S. Albert, P. Schiewe, A. Schöbel, and F. Spühler. LinTim - Integrated Optimization in Public Transportation. Homepage. `https://lintim.net`, 2020.

**25** A. Schiewe, S. Albert, P. Schiewe, A. Schöbel, and F. Spühler. LinTim: An integrated environment for mathematical public transport optimization. Documentation for version 2020.12, 2020. URL: `https://nbn-resolving.org/urn:nbn:de:hbz:386-kluedo-62025`.

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

27 P. Tormos, A. Lova, F. Barber, L. Ingolotti, M. Abril, and M. Salido. A genetic algorithm for railway scheduling problems. In *Metaheuristics for scheduling in industrial and manufacturing applications*, pages 255–276. Springer, 2008.

28 A. van den Heuvel, J. van den Akker, and M. van Kooten. Integrating timetabling and vehicle scheduling in public bus transportation. Technical report, Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, 2008.

## A Local Search

The local search algorithm used here, first described in [19], can be found in Algorithm 2.

**Algorithm 2** Local search using machine learning, as stated in [19].

```
Data: the starting solution currentSolution
currentValue = evaluateByOracle(currentSolution)
while true do
    bestImprovement = $\emptyset$
    bestValue = $\infty$
    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 neighborhood 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
```

## B Dataset information

Figures 11– 14 provide a visualization of the datasets used in this paper.

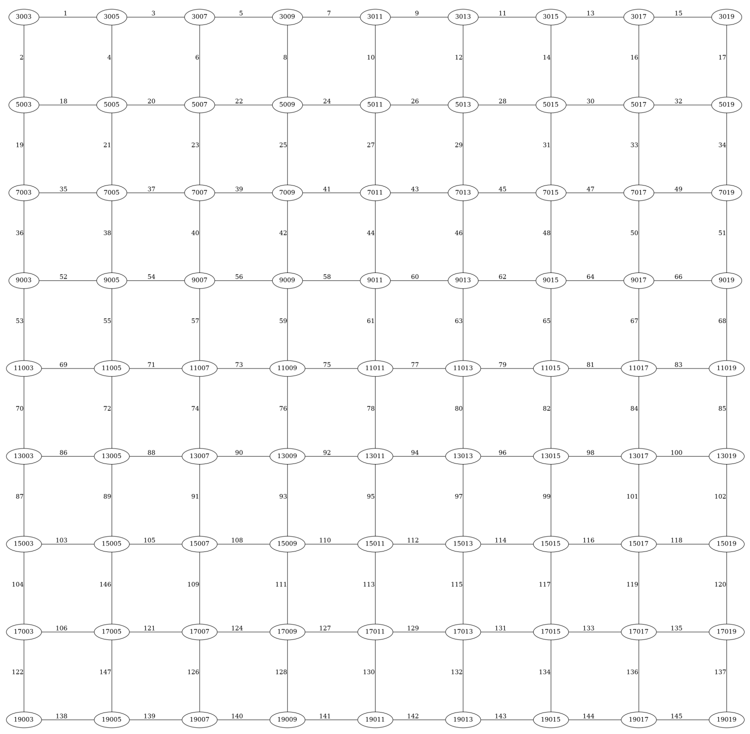


**Figure 11** Infrastructure network of `grid`.

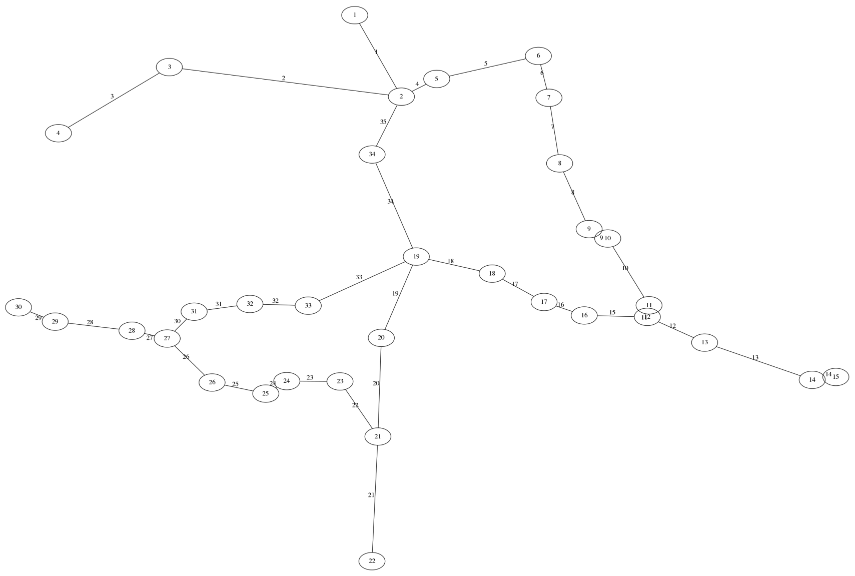


**Figure 12** Infrastructure network of `lowersaxony`.

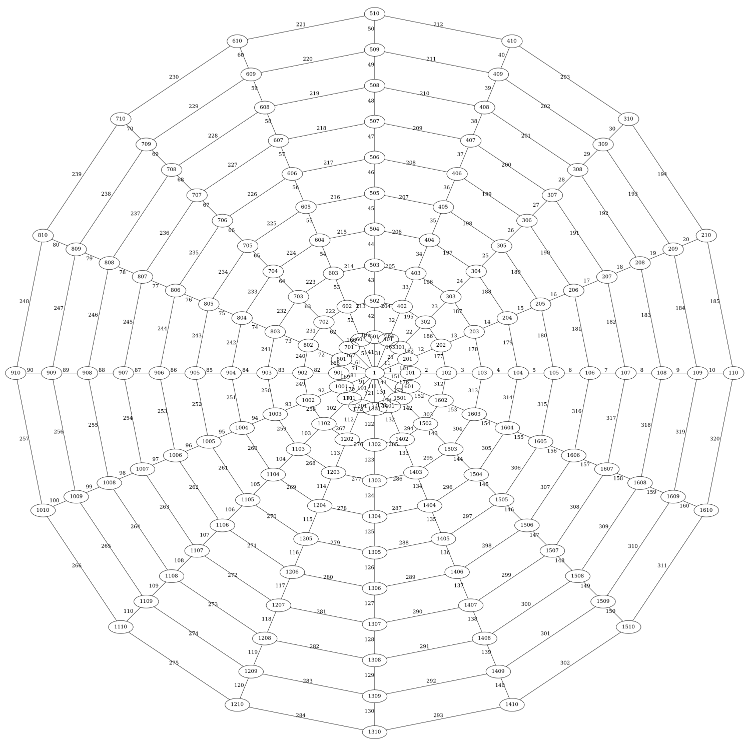


**Figure 13** Infrastructure network of `ring`.

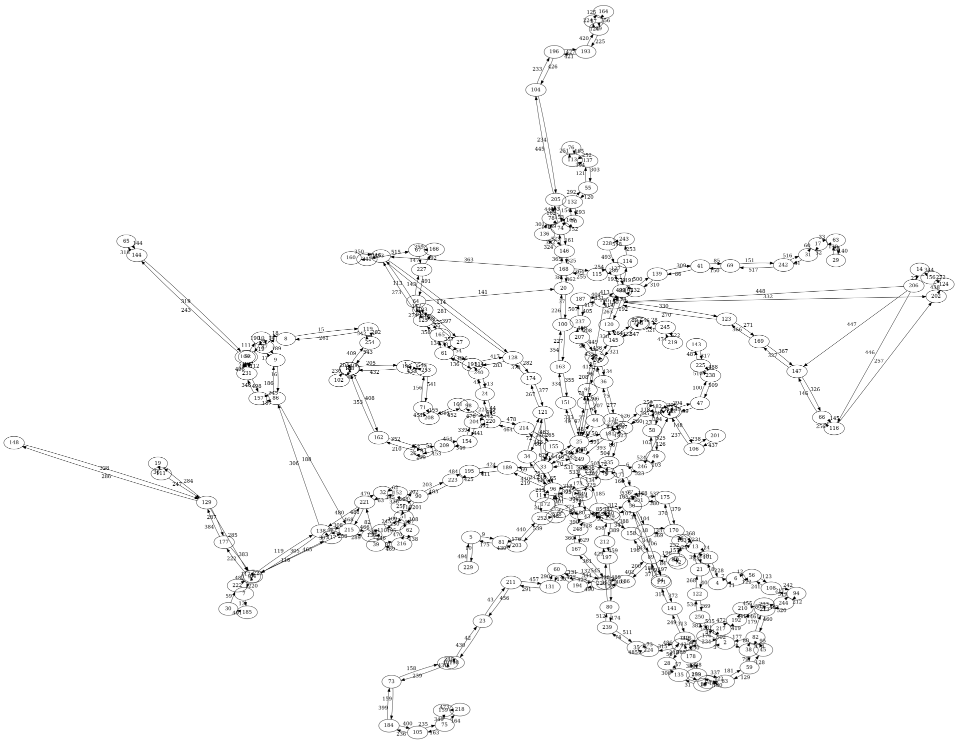


**Figure 14** Infrastructure network of `goevb`.