Finding Robust Timetables for Public Transportation Networks

Anna Pessarrodona Marfà & Tim Seppelt

July 2018

Contents

1. Introduction

The word robustness, when applied to a system, refers to its ability of tolerating perturbations in its input without major effects on its outputs. In our case the systems will be the timetables of public transportation networks, and the perturbations, any kind of delay that might happen. Some of the causes for the delays will be external and unpredictable, such as trees on the track or bad weather, some will be just the consequence of those spreading through the system, with the potentially catastrophic consequences that we have all experienced or seen in the news.

Since there is no way of preventing every kind of imaginable incidence on a track or in a station, we cannot expect the source delays to just disappear. However, we can design systems in a way that they deal with the unavoidable in better ways, and that is exactly what robustness is about. In order to evaluate and improve robustness, though, we first need a model of the public transportation network with all the components that play a role in our problem: we need stations, lines, timetables, passengers, and finally, a more or less benign delay scenario to put it through.

Luckily for us, we are not the first to step onto this tricky field, and most of the mentioned above is already done by a software created by researchers of the university of Göttingen called LinTim[1]. Providing it with passengers and a graph of connected cities, LinTim computes lines, schedules and even yield them to some delay scenario to see how they perform.

To evaluate its ability to cope with delays numerically, we have defined a coefficient of robustness and tried different methods to optimise it, with several degrees of success and some side effects.

[1] See [2] for the LinTim version we have been working with.

2. Theoretical Background

In this section mathematical concepts from public transportation networks modelling will be introduced. Most of them are extracted from [6].

Definition 2.1. A **public transportation network (PTN)** is a graph $G = (V, E)$ given by a set of stops or stations V and a set $E \subseteq V \times V$ of direct connections between them. We assume that a PTN is an undirected graph.

Definition 2.2. A **line** L is a path in the PTN (given by its stations or by its edges).

The **frequency** f_L of a line L says how often service is offered along line L within the (given) planning period I.

A **line concept** is the pair $(\mathcal{L}, f)$, where $\mathcal{L}$ denotes the set of all lines which are operated, the **line plan**, and $f = \{f_L \in \mathbb{N} \mid L \in \mathcal{L}\}$ denotes the set of their respective frequencies.

Definition 2.3. Let a PTN, and a line concept $(\mathcal{L}, f)$ be given.

A **trip** t is the journey of a single vehicle from the start to the end of a line given as path in the PTN through nodes and/or edges. In order to simplify it is assumed that a trip visits a station at most once.

Definition 2.4. Let $L_1, \ldots, L_m$, $m \in \mathbb{N}$ be lines on a PTN $G = (S, C)$.

An **Event-Activity-Network** or **EAN** is a directed graph $\mathcal{N} = (\mathcal{E}, \mathcal{A})$, where the nodes in $\mathcal{E} = \mathcal{E}_{\text{dep}} \sqcup \mathcal{E}_{\text{arr}}$ are *departure* and *arrival events*, respectively, each belonging to a stop from the PTN and a line:

$$\mathcal{E}_{\text{dep}} = \{(s, L, \text{dep}) \in S \times \{L_1, \ldots, L_m\} \times \{\text{dep}\} \mid \exists s_0 \in S : (s, s_0) \in L\}$$
$$\mathcal{E}_{\text{arr}} = \{(s, L, \text{arr}) \in S \times \{L_1, \ldots, L_m\} \times \{\text{arr}\} \mid \exists 0 \in S : (s_0, s) \in L\}$$

The set of arcs (directed edges) $\mathcal{A} = \mathcal{A}_{\text{drive}} \sqcup \mathcal{A}_{\text{wait}} \sqcup \mathcal{A}_{\text{change}} \sqcup \mathcal{A}_{\text{head}}$ consists of so-called **activities**: The **driving**, **waiting**, **changing** and **headway** activities, respectively.

$$\mathcal{A}_{\text{drive}} = \{((s_1, L, \text{dep}), (s_2, L, \text{arr})) \in \mathcal{E}_{\text{dep}} \times \mathcal{E}_{\text{arr}} : (s_1, s_2) \in L\}$$
$$\mathcal{A}_{\text{wait}} = \{((s, L, \text{arr}), (s, L, \text{dep})) \in \mathcal{E}_{\text{arr}} \times \mathcal{E}_{\text{dep}}\}$$
$$\mathcal{A}_{\text{change}} = \{((s, L, \text{arr}), (s, L', \text{dep})) \in \mathcal{E}_{\text{arr}} \times \mathcal{E}_{\text{dep}} : L \neq L'\}$$
$$\mathcal{A}_{\text{head}} = \{((s, L, \text{dep}), (s, L', \text{dep})) \in \mathcal{E}_{\text{dep}} \times \mathcal{E}_{\text{dep}} : L \neq L'\}$$

Driving and waiting activities refer to vehicles of a line driving and waiting, respectively, while changing activities represent customers changing from one vehicle to another in one station. Headway activities model security distances between the departures of different vehicles from the same station, e.g. in train networks.

Definition 2.5. Let $\mathcal{N} = (\mathcal{A}, \mathcal{E})$ be an EAN. For all activities in $\mathcal{A}$, we will set three time restrictions:

- The **fastest possible time** $b_m : \mathcal{A} \longrightarrow \mathbb{N}_0$, which will be fix for every $a \in \mathcal{A}$.
- The **lower bound** $b_0 : \mathcal{A} \longrightarrow \mathbb{N}_0$, which indicates the minimum time that can be assigned to an activity $a \in \mathcal{A}$ while timetabling.
- The **upper bound** $b_1 : \mathcal{A} \longrightarrow \mathbb{N}_0$, which indicates the maximum time that we allow an activity $a \in \mathcal{A}$ to take, including delays.

These lower and upper bounds will represent time constraints considered for the time consumption on each activity, and we will change them in the delay management process.

The fastest possible time, however, will not appear directly represented in the program for technical reasons, cf. [1], but it will be necessary to define the notion of slack.

Definition 2.6. Let $w : \mathcal{A} \longrightarrow \mathbb{N}_0$ be a general weight function. We will use $w(a)$, $a \in \mathcal{A}$ to describe the **number of passengers** in the activity $a \in \mathcal{A}$.

As $\mathcal{A}$ is finite, b_i and w can also be written as vectors with indices in $\mathcal{A}$. Whenever an EAN is referred to as a tuple $(\mathcal{E}, \mathcal{A}, \mathcal{L}, w)$, $\mathcal{E}$ and $\mathcal{A}$ are implicitly meant to be partitioned as above.

Weights could for example be expected passenger loads on the respective activities. Driving and waiting activities are derived directly from the trips on the PTN.

2.1 Timetabling

Definition 2.7. Let $\mathcal{N} = (\mathcal{A}, \mathcal{E})$ be an EAN. A **timetable** $\pi : \mathcal{E} \to \mathbb{N}_0$ is given by (natural) numbers:

- $\pi(v, t, \text{arr}) \in \mathbb{N}\ \forall (v, t, \text{arr}) \in \mathcal{E}_{\text{arr}}$, representing the **arrival times**,
- $\pi(v, t, \text{dep}) \in \mathbb{N}\ \forall (v, t, \text{dep}) \in \mathcal{E}_{\text{dep}}$, representing the **departure times**.

As the set of events is finite, we may refer to a timetable π as a vector in $\mathbb{N}_0^{\#\mathcal{E}}$. In short, we write $\pi_i \in \mathbb{N}_0$ $\forall i \in \mathcal{E}_{\text{arr}} \sqcup \mathcal{E}_{\text{dep}}$.

Definition 2.8. A timetable π is called **feasible** if for all activities $a = (e_0, e_1) \in \mathcal{A}; e_i \in \mathcal{E}$,

$$\pi(e_1) - \pi(e_0) \leq b_1(a) - b_0(a)$$

Definition 2.9. A timetable π is called **periodic** if there exists a period T_L for each line $L \in \mathcal{L}$ such that for every pair of consecutive trips t_0, t_1 in the same direction of line L the following two conditions hold:

(a) $\pi(v, t_0, \text{dep}) = \pi(v, t_1, \text{dep}) + T_L$ for all departure events $(v, t_0, \text{dep}), (v, t_1, \text{dep})$ with $v \in L$

(b) $\pi(v, t_0, \text{arr}) = \pi(v, t, \text{arr}) + T_L$ for all arrival events $(v, t_0, \text{arr}), (v, t_1, \text{arr})$ with $v \in L$

When these conditions are not met, we will refer to those timetables as **aperiodic**. A timetable π is called **periodic with one common period** T, if $T = T_L$ for all lines $L \in \mathcal{L}$.

Definition 2.10. We will refer as **disposition timetable** to the delayed timetable, in contrast with the **nominal timetable**, the original undelayed timetable.

Observation 2.11. The disposition timetable will never be periodic, since delays will not be either.

Definition 2.12. Let $\mathcal{N} = (\mathcal{A}, \mathcal{E})$ be an EAN and π a timetable. Let $a \in \mathcal{A}$ be the activity from event i to event j, and let π_i, π_j be its respective times. We define the **slack time of the activity**, s_a, as:

$$s_a = \pi_j - \pi_i - b_m(a)$$

Where b_m represents the already seen fastest possible time an activity can take.

Optimising robustness is often done by increasing the "buffer time" of the activities in the network. This motivates the following notion.

Definition 2.13. Let $\mathcal{A}$ denote the set of activities of an EAN, $s_a = \pi_j - \pi_i - L_a$ the slack time of the activity $a = (i, j) \in \mathcal{A}$. Then we call $s_T = \sum_{a \in \mathcal{A}} s_a$ the **total slack** of the network.

2.2 Robust optimisation

The problem of finding a timetable given a set of activities with time constraints is modelled by an optimisation problem. However, once we begin to deal with the uncertainty that delays represent, we need to take an adapted approach.

Definition 2.14. Given a general optimisation problem:

$$\begin{aligned} (\mathrm{P}) \quad & \min_x \quad f(x) \\ & \text{s.t.} \quad F(x) \geq 0 \end{aligned}$$

with any objective function f and an arbitrary condition function F, the corresponding **uncertain problem** is given by

$$\begin{aligned} (\mathrm{P}(\xi)) \quad & \min_x \quad f(x, \xi) \\ & \text{s.t.} \quad F(x, \xi) \geq 0 \end{aligned}$$

We will call the uncertain input data ξ **delay scenario** and we will call the set of all scenarios U **uncertainity set** or **scenario set**. We will call $\hat{\xi} \in U$ **nominal scenario**, if for the two problems $f(\cdot) = f(\cdot, \hat{\xi})$ holds.

Definition 2.15. In the case of periodic event scheduling problems (PESP) the solution of the problem (P) or equivalently the solution of $(\mathrm{P}(\hat{\xi}))$ is called **nominal timetable**. For a delay scenario ξ the solution of $(\mathrm{P}(\xi))$ is called **disposition timetable**.

We can consider several kinds of approaches to uncertain problems, which are introduced below. The reader may be pointed to [3] and [5] where more details are provided.

2.2.1 Strict Robustness

In this case, solutions will have to be feasible for all scenarios in the uncertainty set.

Definition 2.16. Given an uncertain optimisation problem:

$$\begin{aligned} (\mathrm{P}(\xi)) \quad & \min_x \quad f(x, \xi) \\ & \text{s.t.} \quad F(x, \xi) \geq 0 \end{aligned}$$

its **strict robust counterpart** with regard to an uncertainty set U is the problem

$$\begin{aligned} (\mathrm{SR}) \quad & \min_x \sup_{\xi \in U} \quad f(x, \xi) \\ & \text{s.t.} \quad F(x, \xi) \geq 0 \quad \forall \xi \in U \end{aligned}$$

Strict robustness is generally not the best approach to robust timetabling because it is too conservative. As delay scenarios can be almost arbitrarily severe taking all of them into account would rule out many desirable solutions.

2.2.2 Light robustness

Definition 2.17. Given a robust optimisation problem with objective function f and constraint function F, both depending on the scenario out of some specified uncertainty set U, the **light robust counterpart** is

$$
\begin{aligned}
(\mathrm{LR}(\delta)) \quad \min_{\gamma} \quad & \bar{\omega}^T \gamma \\
\text{s.t.} \quad & F(x, \hat{\xi}) \geq 0 \\
& f(x, \hat{\xi}) \leq (1+\delta)\bar{z} \\
& F(x, \xi) + \gamma \geq 0 \quad \forall \xi \in U \\
& \gamma \geq 0
\end{aligned}
$$

where $\hat{\xi}$ denotes the nominal scenario, $\bar{z}$ is the solution to (P), $\delta \geq 0$ is a scalar parameter that limits how much can the objective function be apart from the minimum, $\bar{\omega}$ a weight vector.

A light robust solution must thus still satisfy strict feasibility for the nominal scenario and ensure that its quality is not more than a factor of δ apart from the optimum.

2.2.3 A measure for robustness

There are several functions that can be used as a robustness coefficient, which take into account parameters such as average travelling time or connections missed. For simplicity, we have taken average travelling time as the only delay-depending parameter:

Definition 2.18. Given a nominal timetable and a disposition timetable to a delay scenario, we will call the **robustness** or **robustness coefficient** $R \in [1, \infty)$ of a timetable the quotient of the average travelling time of the disposition timetable by the average travelling time of the nominal timetable.

Analogously, for more detailed analysis one may consider the following coefficient:

Definition 2.19. Let $d(a) \in \mathbb{N}_0$ denote the source delay of an activity $a \in \mathcal{A}$ in a given delay scenario. Let $\Theta > 0$ denote the nominal average travelling time. Then we define the **robustness coefficient w.r.t. source delays** R_s as

$$R_s := \frac{1}{\Theta} \frac{\sum_{a \in \mathcal{A}} d(a) w(a)}{\sum_{a \in \mathcal{A}} w(a)},$$

where $w(a)$ denotes the number of passengers on the activity a.

Note, that for comparability Θ and $d(a), a \in \mathcal{A}$ must have the same unit and that R_s is not necessarily in $[1, \infty)$. Generally, only $R_s \geq 0$ holds.

3. Modelling Delays

When referring to delays, it is important to distinguish two types of them, according to their origin:

- **Source delays**, caused by circumstances which are external to the PTN such as damaged tracks, incidences in stations, bad weather or other incidences. They are unpredictable.
- **Propagated delays**, caused by the propagation of those source delays across the system. Examples of this are trains that might have to wait for other trains, overbooking of tracks or delayed departures because of minimum safety distance between trains, i.e. headways. We can compute them using LinTim and thus we can expect to reduce them.

We will model source delays by assuming probability distributions on uncertainty sets, i.e. by turning uncertainty sets into probability spaces. Propagated delays depend on the delay management strategy of the system, and we will analyse them further in the following chapters.

In this work we have only considered delays on driving activities, and not on events, nor on other types of activities. The reason to do so is to facilitate the comparison of the results from different random variables or different PTNs.

3.1 Model

We have worked with three different models for source delays, two of them using uniform distributions for the random variable, and one a combination of binomial and exponential distributions.

3.1.1 Albert's uncertainty sets

In [3], Albert introduces two possible uncertainty sets, defined as follow:

- $U_1(s), s \geq 0$, consists of all delay scenarios where all driving activities can be delayed by a factor in $[0, s]$.
- $U_2(s, k), s \geq 0, k \in \mathbb{N}_0$, consists of all delay scenarios where not more than k driving activities can be delayed by a factor in $[0, s]$.

Note that, $U_2(s, k) \subseteq U_1(s)\ \forall s \geq 0$ and $U_2(s, k) = U_1(s)$ for all $k \geq \#\mathcal{A}_{\text{drive}}$. On both sets Albert assumes an uniform distribution. We implemented the scenario generators for both sets as `ScAlbertU1` and `ScAlbertU2`, cf. section 4.2.

3.1.2 The TreeOnTrack model

This model describes source delays in the following way: by default only driving activities can be delayed. The number of possibly delayed activities, X, is modelled through a binomial distribution with parameters (n, p), with n representing the total number of driving activities in the EAN, and p, the probability of each single activity to be possibly delayed:

$$X = \#\text{ of delayed activities} \sim Bin(n, p)$$

For a possibly delayed activity $a \in \mathcal{A}$ the length of the delay (in seconds), Y_a, is modelled by an exponential distribution with the following density function (β parametrization)

$$f_\beta(x) = \begin{cases} \frac{1}{\beta} \exp\left(-\frac{x}{\beta}\right), & x \geq 0 \\ 0, & \text{otherwise} \end{cases}.$$

All random variables $X, Y_a, a \in \mathcal{A}$, are assumed to be pairwise independent.

This model is implemented as `TreeOnTrackScenarioGenerator`.

Observation 3.1. Let $X \sim Bin(n,p), Y_i \sim Exp(\beta), i \in \mathbb{N}$, pairwise independent. The total amount of delay Z is defined as $Z = \sum_{i=1}^{X} Y_i$. It has a mean of $E(Z) = np\beta$ and a standard deviation $\sigma(Z) = \sqrt{V(Z)} = \beta\sqrt{np(2-p)}$.

$$\begin{aligned}
EZ &= E\left(\sum_{i=1}^{X} Y_i \mathbf{1}_{\{0 \leq X \leq n\}}\right) = E\left(\sum_{i=1}^{X} Y_i \sum_{k=0}^{n} \mathbf{1}_{\{X=k\}}\right) = \sum_{k=0}^{n} E\left(\mathbf{1}_{\{X=k\}} \sum_{i=1}^{k} Y_i\right) \\
&= \sum_{k=0}^{n} E\mathbf{1}_{\{X=k\}} E\left(\sum_{i=1}^{k} Y_i\right) = EY_1 E\left(\sum_{k=0}^{n} k\mathbf{1}_{\{X=k\}}\right) = EY_1 EX = np\beta \\
EZ^2 &= E\left(\sum_{i=1}^{X} Y_i^2 + \sum_{i \neq j}^{X} Y_i Y_j\right) = \sum_{k=0}^{n} E\left(\left(\sum_{i=1}^{k} Y_i^2 + \sum_{i \neq j}^{k} Y_i Y_j\right) \mathbf{1}_{\{X=k\}}\right) \\
&= \sum_{k=0}^{n} \left(E\mathbf{1}_{\{X=k\}} \left(\sum_{i=1}^{k} EY_i^2 + \sum_{i \neq j}^{k} EY_i Y_j\right)\right) = \sum_{k=0}^{n} E\mathbf{1}_{\{X=k\}} \left(kEY_1^2 + (k^2-k)(EY_1)^2\right) \\
&= (EY_1)^2 EX^2 + VY_1 EX = \beta^2 np(2-p+np)
\end{aligned}$$

4. Evaluating Robustness

4.1 Model

As said in the introduction, the concept of robustness refers to the ability of a certain timetable to minimize the effects of source delays on the average travelling time of the passengers.

In order to measure the robustness of a timetable we will conduct the following probabilistic experiment: For a set of randomly generated source delays (delay scenarios), a disposition timetable and statistical data is computed. Finally the data points describing the single iterations are summarized to the robustness properties of the timetable.

Note that the robustness coefficient R not only depends on the delay scenario, e.g. p and β, but also on other network characteristics. Hence, the robustness coefficients of different networks, e.g. `athens` and `bahn-01`, cannot be compared without additional effort.

4.2 Implementation

LinTim provides a variety of components and options for computing and analysing the effect which source delays have on a given timetable, cf. [1]. Therefore our approach included firstly to set up a library which communicates with LinTim components and secondly to configure and enhance parts of the given functionality. In order to make the library flexible for further development we used an object oriented plugin-in architecture.

The central class of the robustness evaluation is `RobustnessEvaluator`[1]. Its functionality is described in the section above and schematically in Figure 4.1. The evaluation consists of three major parts which are modelled by the following components:

- A `ScenarioGenerator` generates a series of delay scenarios. Its functionality is similar to LinTim's `dm-delays`. Python's generator feature is used for iteratively generating delay files, storing them in the respective LinTim directories and handing over properties of the delay scenarios to the `Statistician`.
- A `DelayManager` takes care of computing a disposition timetable. It wraps around LinTim's `dm-disposition-timetable`. We did not make any adjustments at this part of the evaluation process.
- A `Statistician` fetches, stores and analyses statistical data about the delay scenarios and the disposition timetables. For this we used the output of LinTim's `dm-disposition-timetable-evaluate`. The `Statistician` is called before the first iteration of the evaluation for collecting data about the nominal timetable, in every iteration for collecting data about the disposition timetables and after the last iteration for returning a summary of the statistical data.

These classes do not implement the respective steps of the evaluation and must be understood as placeholders. We provide several implementations, i.e. inheriting classes, which can of course be extended as well.

The TreeOnTrack model and functionality for measuring the robustness coefficient R, which are described in subsection 3.1.2 and in Definition 2.18, are implemented in the following classes:

- `TreeOnTrackScenarioGenerator(`p`,`β`)` generates a series of TreeOnTrack distributed delay scenarios with parameters p and β.

[1] All classes which are related to robustness evaluation are located in the `robtim.eval` package.

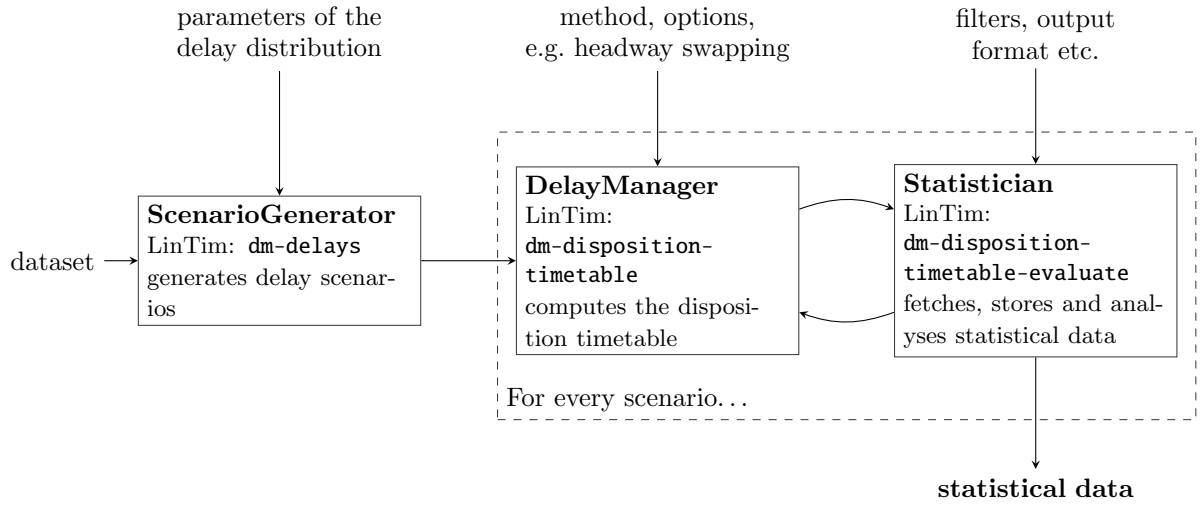


Figure 4.1: Overview of the robustness evaluation. The boxes represent software components, all other nodes represent input or output data.

- `CoefficientStatistician(['dm_time_average'])` computes the robustness coefficient for the set of delay scenarios.

For further details see the attached documentation.

4.3 Results

As we can see in the graphs in Figure 4.2 from different datasets generated using TreeOnTrack delay scenarios, there is a direct correlation between more delays and longer travelling times.

4.3.1 p-β-graph

From now on, we will use the TreeOnTrack model in all our delay studies. The previous graphs suggest that robustness could be analysed not on p and β as separated parameters, but in terms of $p\beta = \text{ct}$.

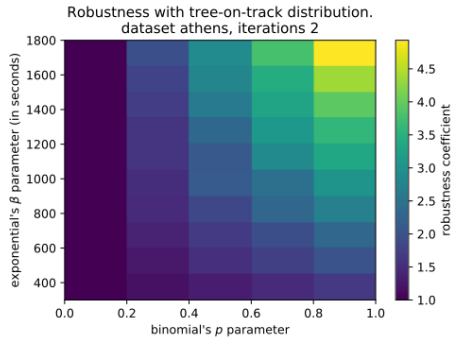
In Figure 4.3 we can see that robustness remains constant when $p\beta$ does. We can also see that, in this case, lower values of p cause a higher dispersion of the points. Extreme travelling times occur only with low p (and high β) but are rare. Instead higher values of p behave in the opposite way: the travelling time is estimated well by the average of the delayed travelling times (orange points in the graphs).

4.3.2 Behaviour of a timetable in many iterations

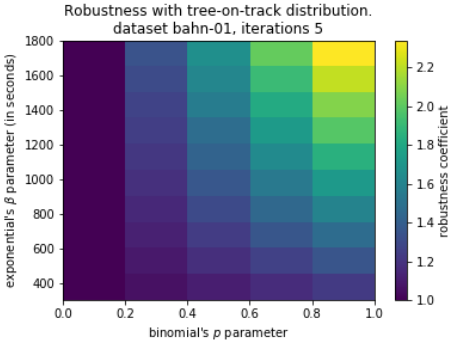
Due to the limitation of computation time this work uses mostly smaller samples, i.e. mostly less than 20 iterations for an evaluation or an optimisation. In order to analyse the behaviour of a timetable in a larger number of iterations, we chose a specific setting and repeated the evaluation of robustness more times.

More precisely, we used the dataset `bahn-01` with a timetable computed by `MATCH`, cf. subsection 5.3.2. The delay scenarios were computed with the TreeOnTrack model with parameters $p = 0.5$, $\beta = 900$. We used 5000 iterations.

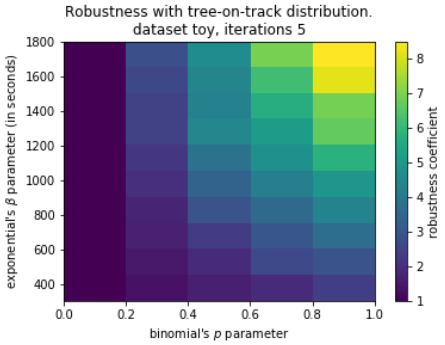
The EAN of the dataset contains 9222 driving activities, which have been potentially affected by delays. The nominal average travelling time in this setting is 26325 s, whereas the average of the delayed average travelling times is 35255 s. Thus, the average robustness coefficient is $R = 1.3992$. Taking Figure 4.2 into account, this



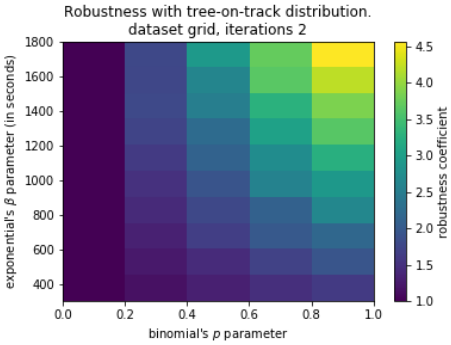
(a) dataset `athens`



(b) dataset `bahn-01`

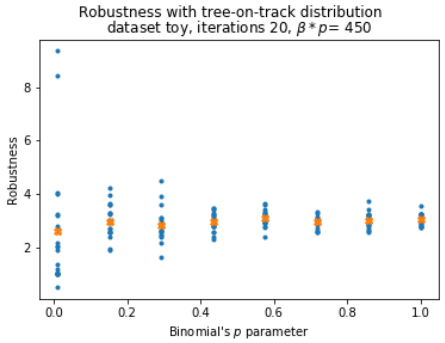


(c) dataset `toy`

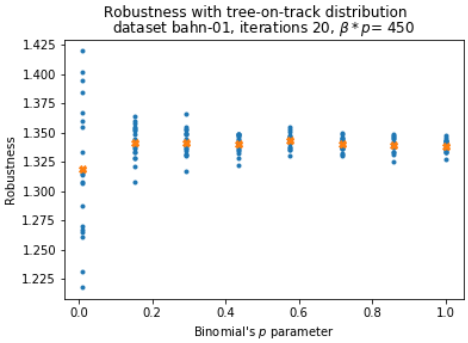


(d) dataset `grid`

Figure 4.2: Comparison of the influence of p and β on the robustness of four different datasets



(a) dataset `toy`



(b) dataset `bahn-01`

Figure 4.3: Robustness with $p\beta$ constant

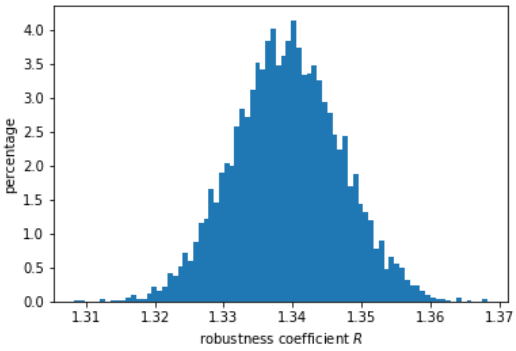


Figure 4.4: Histogram of robustness coefficients in many iterations. Dataset `bahn-01`.

is not surprising. The large sample allows a closer look on the distribution of the robustness coefficients of the iterations. The smallest measured robustness coefficient is 1.3083, the largest coefficient is 1.3682. The standard deviation of the robustness coefficients is 0.0077, i.e. 0.5794 % of the average. A histogram of the robustness coefficients can be found in Figure 4.4.

Recalling the theoretical properties of the delay distribution may now be interesting for answering the following question: Which impact does the network have on the distribution of delays?

Note that the values for the first random variable are the theoretical estimations computed using Observation 3.1, while the next two columns are collected empirically.

	total source delays	average travelling time	robustness coefficient
mean μ	4149900 s	35255 s	1.3992
standard deviation σ	74849 s	204 s	0.0077
coefficient of variation σ/μ	1.8936 %	0.5794 %	0.5794 %
minimal value $v_{\min}$		34442 s	1.3083
... relative to the mean $v_{\min}/\mu$		97.69 %	97.69 %
maximal value $v_{\max}$		36019 s	1.3682
... relative to the mean $v_{\max}/\mu$		102.17 %	102.17 %
span between minimal and maximal value $v_{\max} - v_{\min}$		1577 s	0.0599
... relative to the mean $(v_{\max} - v_{\min})/\mu$		4.47 %	4.47 %

Clearly, the robustness coefficient (and the average travelling time) spread much less than the source delays. The network has apparently the interesting property that it reduces the deviation of input perturbations compared to the output. Thus, the average travelling times can be predicted better than the source delays.

5. Optimising Robustness

5.1 Model

Increasing the robustness of a timetable can be done in many different ways, and even the objective functions used for this optimisation are not always the same, cf. section 2.2. This work focuses on modifying the lower bounds of the activities in an EAN instead of directly working on the timetabling method. Therefore every optimisation step consists mainly of modifying the EAN in a way which increases robustness. The objective function used is the robustness coefficient R as defined in Definition 2.18.

5.2 Implementation

In order to provide a general and useful library for broad variety of optimisation techniques we used the same approach as we did for the `RobustnessEvaluator`, cf. section 4.2. Contrary to the evaluation of robustness the optimisation is mostly not covered by LinTim's components.

The central class of the robustness optimisation is `RobustnessOptimiser`[1]. Its functionality is described schematically in Figure 5.1. The optimisation consists of three major parts which are modelled by the following components:

- An `EANGenerator` generates a series of EANs, e.g. manipulates the lower bounds for the activities. The robustness of the related timetable should increase in every step.
- A `Timetabler` behaves like the LinTim components `tim-timetable` and `ro-rollout`. It computes an aperiodic timetable using the previously generated EAN.
- A `Supervisor` provides a stop criterion and may collect statistical data. It is asked whether the optimisation shall be interrupted in every iteration and can collect statistical data about the current EAN at this point. When the optimisation is finished the `Supervisor` is asked for a report about the optimisation, e.g. the collected statistical data.

These classes do not implement the respective steps of the optimisation and must be understood as placeholders. We provide several implementations, i.e. inheriting classes, which can of course be extended as well. For more details see the attached documentation.

In order to apply the results of chapter 4 one has to use a `Supervisor` which tracks the robustness of the timetables, e.g. `MatrixRobustnessSupervisor` or `RobustnessSupervisor`. In this case an objective value for the robustness can be set as stop criterion.

5.3 Results

5.3.1 Comparison of EAN Generators

In this section three fundamentally different approaches of adding slack to a system are compared. The simplest method adds a constant amount of slack to all activities. A more sophisticated method redistributes the slack among the activities but keeps the total amount of slack in the system constant. In order to be able to compare

[1] All classes which are related to robustness optimisation are located in the `robtim.opt` package.

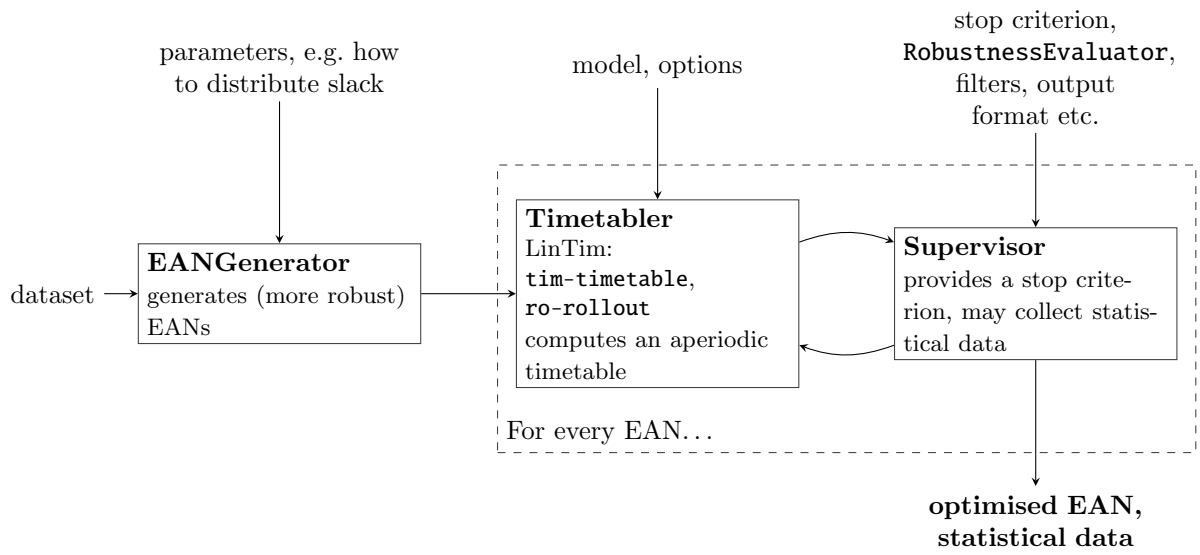


Figure 5.1: Overview of the robustness optimisation. The boxes represent software components, all other nodes represent input or output data.

the results of different EAN generators we kept the other parameters of the optimisation procedure constant, cf. section 5.2. This included the timetabling method as well as the parameters of the robustness evaluation (timetabling method `MATCH`, TreeOnTrack model, p, $\beta = \text{const}$).

Adding an increasing amount of slack to all activities

Aiming at implementing a basic method for increasing the robustness of a timetabling, i.e. of the underlying EAN, we started with adding a constant amount of slack time to all activities in the system. This was done by simultaneously increasing the lower and upper bounds for the activities.

More precisely, in every step $k \in \mathbb{N}_0$ a constant amount of slack $\sigma \in \mathbb{N}$ is added to all non-zero activities $a \in \mathcal{A}$, i.e.

$$s_a^{(k)} = \begin{cases} k\sigma, & w(a) > 0 \\ 0, & \text{otherwise} \end{cases}$$

This method is implemented as `IncreasingSlackEANGenerator`.

Figure 5.2 illustrates the results. In every iteration $\sigma = 5$ min of slack time were added to all non-zero activities. The positive effect on the robustness coefficient is obvious. On the other hand the average travelling time increases from 30 000 to 100 000 s.

Redistributing the slack

Another strategy for increasing the robustness of a timetable is to at slack only to the most important activities in the network. Intuitively this seems to be a smarter technique since the nominal travelling times are only increased selectively. For doing this, we identified the important activities as those with greater numbers of passengers. In every iteration of the optimisation the number of activities which are important enough to receive slack is reduced. Keeping the iterative design this EAN generator works in the following way:

Let s_T be the fixed total amount of slack, $n > 0$ the step size. Let all activities be sorted by their number of passengers in reverse order. Choose indices according to this order such that the first activity has the most passengers etc. Let $\mathcal{A}_{>0} := \{a \in \mathcal{A} \mid w(a) > 0\}$ denote the set of all activities with at least one passenger.

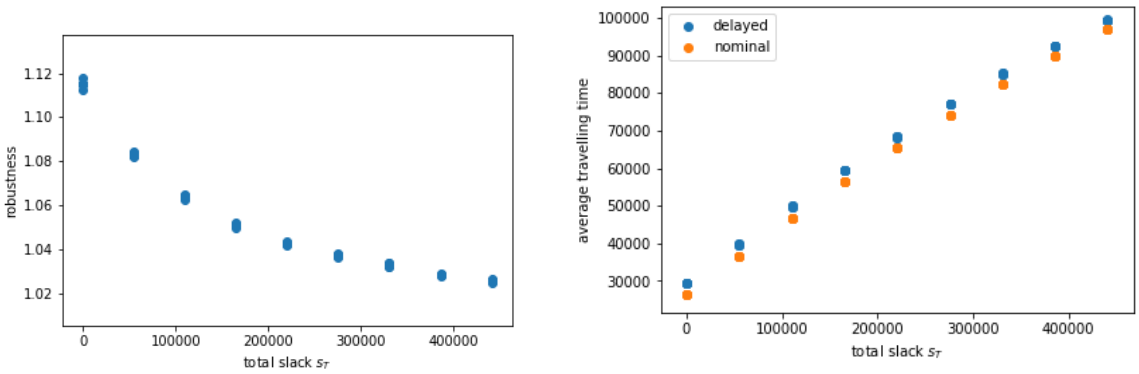


Figure 5.2: Adding an increasing amount of slack to all activities of the dataset `bahn-01` with $\sigma = 5, p = 0.5, \beta = 300$, 9 optimisation iterations, each with 5 evaluation iterations.

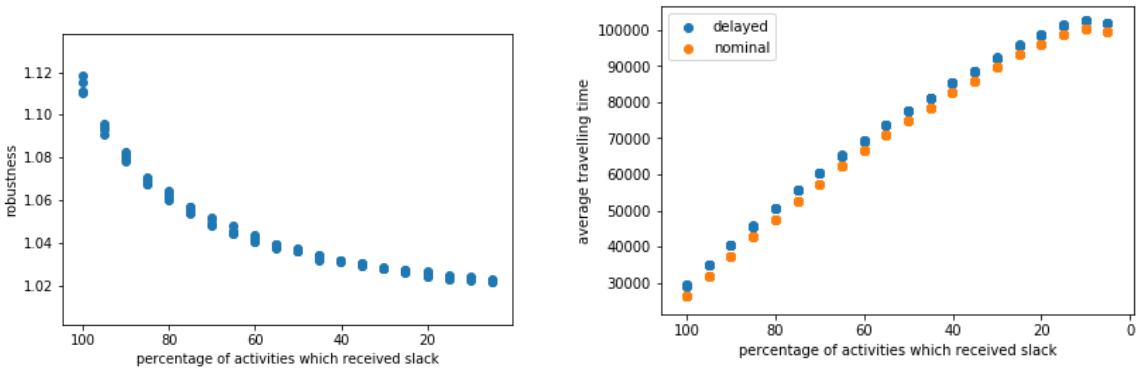


Figure 5.3: Redistribution of slack on the dataset `bahn-01` with $s_T = 10\,000$, $n = 551 = 5\%$, $p = 0.5$, $\beta = 300$, 20 optimisation iterations, each with 5 evaluation iterations.

In every iteration $0 \leq k \leq \frac{|\mathcal{A}_{>0}|-1}{n}$ the slack is distributed according to

$$s_a^{(k)} = \begin{cases} s_T w(a) \left(\sum_{j=1}^{|\mathcal{A}_{>0}|-kn} w(j)\right)^{-1}, & a = 1, \ldots, |\mathcal{A}_{>0}| - kn \\ 0, & \text{otherwise} \end{cases}$$

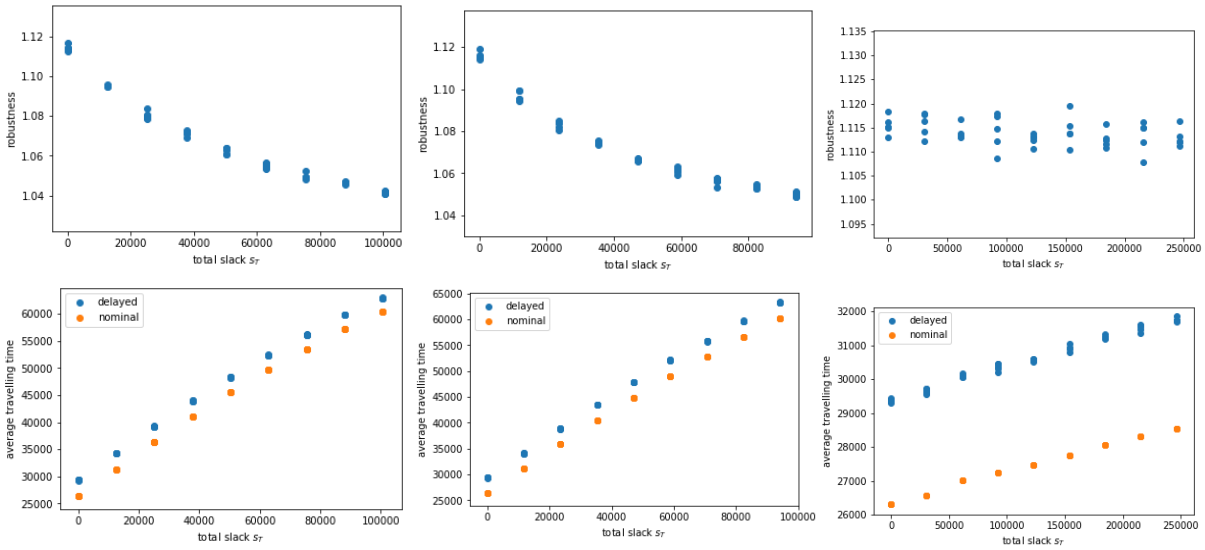
The total amount of slack is indeed constant:

$$s_T^{(k)} = \sum_{a \in \mathcal{A}} s_a^{(k)} = \sum_{a=1}^{|\mathcal{A}_{>0}|-kn} s_a^{(k)} = s_T \left(\sum_{a=1}^{|\mathcal{A}_{>0}|-kn} w(a)\right)^{-1} \left(\sum_{a=1}^{|\mathcal{A}_{>0}|-kn} w(a)\right) = s_T, \ \forall 0 \leq k \leq \frac{|\mathcal{A}_{>0}| - 1}{n}$$

Lower and upper bounds are increased simultaneously by $s_a^{(k)}$ for all a. This method is implemented as `GiveTheRichEANGenerator`.

Exemplarily, the results which we achieved using this technique are shown in Figure 5.3. It is obvious that even though the robustness of the timetables increases significantly from 1.12 to 1.02 the average travelling time is increasing as well from 26 325 s to about 100 000 s. In other words, the robustness of the system was improved by 9.8 % while the average travelling was increased by about 280 %. Of course, this cannot be a suitable result of an integral robustness optimisation.

On the other hand, the figure shows that the robustness coefficients decrease much faster than the average travelling time increases. This leads to the problem of finding a trade-off between robust and fast timetables, which is discussed in chapter 6.



(a) driving activities (b) waiting activities (c) changing activities

Figure 5.4: Impact of slack time for activities of a certain type on the robustness of the network. Dataset `bahn-01` with `IncreasingSlackEANGenerator`, $\sigma = 5, p = 0.5, \beta = 300$, 9 optimisation iterations, each with 5 evaluation iterations. The upper graphs show the robustness and the lower graphs the average travelling times of the networks compared to s_T.

Identifying important activities

In contrary to increasing or redistributing the slack among driving activities, it is reasonable to change the lower and upper bounds of other activities as well. In this section we try to analyse the impact of slack time for all types of activities on the robustness of the network. By doing this we aim at keeping the intervention as little as possible, since optimising the robustness of a network must on the long run not only take into account the robustness coefficient but also the average travelling time.

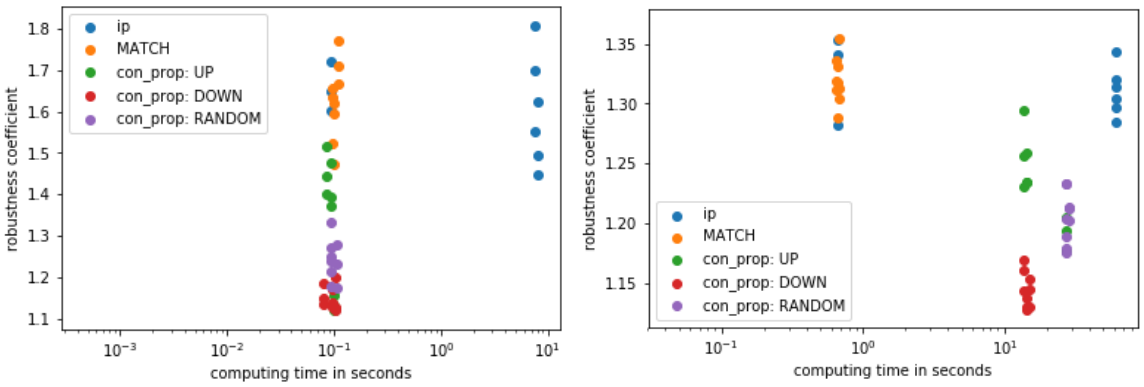
Exemplarily, our results are shown in Figure 5.4. The analysis of the impact of slack time for driving, waiting and changing activities on the robustness of the network shows that only driving and waiting activities influence the robustness. Adding slack to changing activities does not produce any significant results. Waiting activities have a slightly more intense influence, i.e. the robustness coefficient decreases slightly faster as the average travelling time increases in comparison to driving activities. Thus, when developing more advanced techniques for identifying important activities in the described sense one can with a clear conscience focus on driving and waiting activities.

5.3.2 Comparison of different timetabling techniques

Apart from adding slack to specific activities one may consider to do the timetabling in a more robust way. This approach does not focus on the input data for the timetabling step but on this step itself. The general timetabling problem is NP-hard, cf. [6]. Therefore, the optimal solution is in most cases out of reach. Heuristics and simplifications are used in order to obtain a procedure which is feasible for real world applications. That is why, when comparing different timetabling techniques, one must not only take the achieved robustness but also the computing time into account.

LinTim provides a variety of different timetabling methods, cf. [1]:

- `IP` solves an integer programming model of the periodic timetabling problem.



(a) dataset `toy`

(b) dataset `athens`

Figure 5.5: Comparison of different timetabling techniques on two datasets. $p = 0.5, \beta = 300$, 3 iteration for each timetabling method, each with 3 iterations of robustness evaluation

- `MATCH` computes a timetable using the matching-merge heuristic, cf. [4].
- `con_prop` computes a timetable by propagating constraints. Times of events are fixed and propagated through the network. Backtracking is used to fix times differently when running out of feasible solutions. The algorithm can fix event times in different ways:
  - `UP` chooses tight event times.
  - `DOWN` relaxes the event times as much as possible.
  - `RANDOM` chooses event times randomly from the range of feasible times.

Figure 5.5 shows the results of the comparison. Unfortunately, for more complex datasets like `bahn-01` not all algorithms terminate in appropriate time. Therefore the two smaller datasets `toy` and `athens` are compared. The figure illustrates that constraint propagation leads to the most robust timetables while using moderate amount of computing time. `MATCH` is the fastest algorithm while providing the least robust timetables. In contrary to that, the integer programming model takes under some circumstances the longest time without guaranteeing necessarily better robustness results. We can conclude that constraint propagation appears as the best strategy.

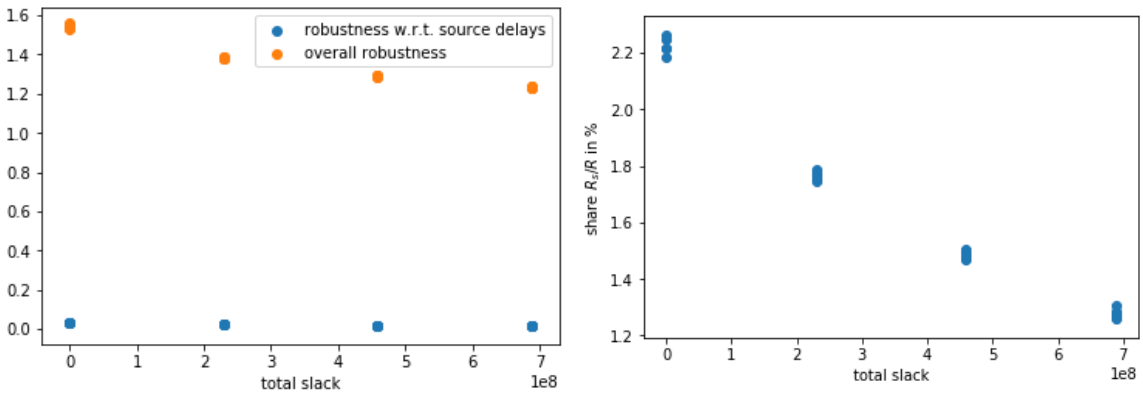
5.3.3 Impact of source delays in comparison to propagated delays

Which effect source delays have in comparison to propagated delays not only may be of interest when analysing the quality of a delay management method, but reveals also interesting properties of the network and the delay scenario. Especially when optimising a timetable one could look at source and propagated delays separately and see at which point in the delay management they take effect.

In this section we consider the previously defined robustness coefficient R and the coefficient R_s, which measures the robustness only with respect to source delays, cf. Definition 2.18, Definition 2.19.

We collected data about source and propagated delays for the dataset `bahn-01` and the `TreeOnTrack` model. The results are shown in Figure 5.6. Interestingly, source delays have only a very little effect on the total robustness. For example, in the last considered optimisation step the overall robustness was $R \approx 1.2346$ and the robustness w.r.t. source delays was $R_s \approx 0.0158$.[2]

[2] The values of R and R_s are averages over all evaluation iterations for the last optimisation step.



(a) robustness coefficients R and R_s

(b) ratio R_s/R in %

Figure 5.6: Impact of source delays in comparison to propagated delays. Dataset `bahn-01`, `TreeOnTrack` model, $p = 0.8, \beta = 900$, `IncreasingSlackEANGenerator`, $\sigma = 5$, 4 optimisation iterations each with 5 evaluation iterations

Their share R_s/R is maximally 2.3 % and decreases rapidly during optimisation. This shows that the optimisation (in this case: `IncreasingSlackEANGenerator`, $\sigma = 5$ min) is highly effective limiting the impact of source delays on the robustness. Furthermore, this suggests that focusing on the limitation of delay propagation could be fruitful.

The coefficients decrease differently fast during optimisation. In the considered 4 optimisation steps and when looking at minimal and maximal values R decreases by 21 %, R_s by 55 % and the ratio R_s/R by 44 %.

This underlines that the used optimisation method produces primarily (1) more robust timetables with respect to source delays, (2) timetables which are proportionally less effected by source delays than by propagated delays.

6. Finding a trade-off

In the last chapters we showed that more robust networks can be generated with one of the described methods. At the same time, the analysis suggested that average travelling times increased in a great extent while optimising. In this section we will describe a method for measuring the quality of an optimisation technique and discuss possible trade-offs.

During optimisation the timetables are modified. Therefore one must analyse the relation between the two main objective values, robustness and average travelling time, for the timetables generated in each iteration. Let $n = 1, \ldots, N$ denote the number of the iterations of the optimisation. Let $R(n)$ denote the robustness coefficient in the n-th iteration of the optimisation. Let $\Theta(n)$ denote the nominal average travelling time in the n-th iteration and $\Theta(0)$ accordingly the nominal average travelling time before optimisation. Then we can define

$$T(n) := \frac{\Theta(n)}{\Theta(0)},\ n = 1, \ldots, N$$

i.e. the ratio in which the nominal average travelling time increases until the n-th iteration of the optimisation. In order to look for a trade-off one may consider for $\lambda \in [0, 1],\ n = 1, \ldots, N$ the generalized objective function

$$\mu_\lambda(n) := \lambda R(n) + (1 - \lambda)T(n)$$

In this way, we obtain for every $\lambda \in [0, 1]$ an optimisation problem

$$\min_{1 \le n \le N} \mu_\lambda(n) = \min_{1 \le n \le N} (\lambda R(n) + (1 - \lambda)T(n)) \qquad (*)$$

which takes the robustness as well as the average travelling times into account. The parameter λ specifies how important the objective robustness is in contrary to the other.

It is obvious that for an optimisation method which increases the average travelling time, e.g. one of the methods described in chapter 5, the solution of $(*)$ has the following form: $n = 1$, if $\lambda = 0$; $n = N$, if $\lambda = 1$ and $1 < n < N$ for $\lambda \in (0, 1)$. Of course, only $1 < n < N$ represents an interesting result since in the two other cases either optimisation is not worth trying at all or an ideal timetable would not take average travelling times into account.

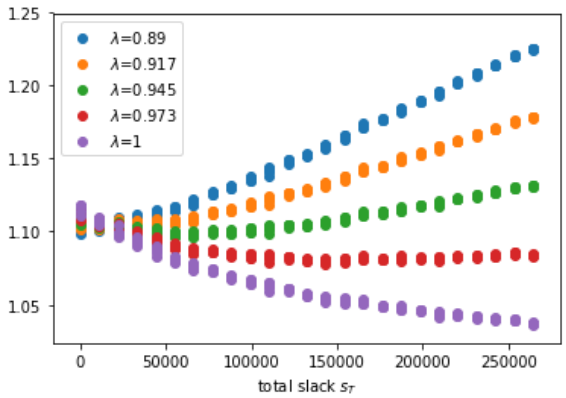
The interval $I \subseteq [0, 1]$, such that for all $\lambda \in I$ the problem $(*)$ has a non-trivial solution, may be considered as a measure for the quality of the optimisation method.

Applying this approach to real data leads to the following results: Figure 6.1 shows that for the two compared optimisation methods $(*)$ has only with great λ non-trivial solutions.[1] The following table allows a closer look at the data. Method 1 refers to adding slack to all activities, Method 2 to redistributing the slack. MIN and MAX mean that the solution of $(*)$ is trivial with $n = 1$ or $n = N$.

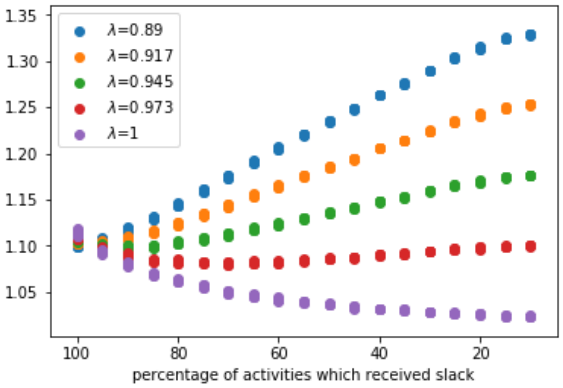
λ	0.0	$\cdots$	0.89	0.901	0.912	0.923	0.934	0.945	0.956	0.967	0.978	0.989	1.0
Method 1	MIN	$\cdots$	MIN	1.1e6	1.1e6	2.2e6	5.5e6	6.6e6	6.6e6	14.3e6	20.9e6	MAX	MAX
Method 2	MIN	$\cdots$	MIN	MIN	95%	95%	90%	90%	90%	75%	65%	MAX	MAX

Both rows show the same monotonic behaviour. The interval with non-trivial solution for the first method is slightly bigger and its minimum is smaller. Thus, this method can under these circumstances be used for optimising robustness when $\lambda \ge 0.901$. Adding slack to all activities appears in this case to be a better optimisation method than redistributing the slack.

[1] The reader may excuse our notational inaccuracy when using the EAN generators' specific properties s_T and the percertange of activities which received slack instead of the iteration index n on the horizontal axis.



(a) adding slack to all activities, $\sigma = 5$, cf. Figure 5.2



(b) redistributing slack, $s_T = 10\,000$, $n = 551 = 5\%$, cf. Figure 5.3

Figure 6.1: Trade-off of robustness and average travelling time, dataset `bahn-01`, $p = 0.5, \beta = 300$, 24-25 optimisation iterations, each with 5 evaluation iterations.

Generally both methods do not appear to be very useful when a trade-off objective is used instead of the robustness objective. This opens the door to further investigations on other methods of improving robustness while constraining the increase of average travelling times.

7. Conclusion

In this project we have seen that it is effectively possible to improve the robustness of a timetable, this is, to reduce the impact of initial delays on the global system of transportation.

We have tried different approaches to this goal. Some of them succeeded, some of them did not but gave us interesting results and ideas on how the work could be continued. Trying different LinTim algorithms gave us better and worse timetables, as well as different computation times. A strategy which definitely improved robustness when taken as sole objective was adding slack time and redistributing it among the connections according to their relevance.

However, this option increased average travelling times dramatically, and it made obvious that robustness, even though it is a desirable feature, cannot be the only one taken into account when designing a timetable.

This considered, we tried to find a trade-off between robustness and nominal travelling time. This trade-off is hard to achieve but necessary since thousands of people are affected by delay management in their everyday lives.

Nevertheless, we hope that our work gives an insight on the complexity of the original problem and inspiration for a possible continuation of the investigations.

List of Figures

Bibliography

[1] Anita Schöbel et al. *LinTim. An integrated environment for mathematical public transport optimization. Documentation*. 2018.

[2] Anita Schöbel et. al. *LinTim. Revision eb286858*. 2018. URL: https://gitlab.gwdg.de/LinTim/LinTim/commit/ef000c8b3926e8235601e7299e340314eb286858.

[3] Sebastian Albert. *Evaluation of Robust Timetables*. 2012.

[4] Julius Pätzold and Anita Schöbel. "A Matching Approach for Periodic Timetabling". In: *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*. Ed. by Marc Goerigk and Renato Werneck. Vol. 54. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016, 1:1–1:15. ISBN: 978-3-95977-021-7. DOI: 10.4230/OASIcs.ATMOS.2016.1. URL: http://drops.dagstuhl.de/opus/volltexte/2016/6525.

[5] Michael Schachtebeck. "Delay Management in Public Transportation: Capacities, Robustness, and Integration". PhD thesis. 2009.

[6] Anita Schöbel. *Lecture Notes on Optimization Models in Public Transportation*. 2017.