Project Number 288094

eCOMPASS

eCO-friendly urban Multi-modal route PlANning Services for mobile uSers

STREP
Funded by EC, INFSO-G4(ICT for Transport) under FP7

eCOMPASS – TR – 060

Delay-Robust Journeys in Timetable Networks with Minimum Expected Arrival Time

Julian Dibbelt, Ben Strasser, and Dorothea Wagner

June 2014

Delay-Robust Journeys in Timetable Networks with Minimum Expected Arrival Time*

Julian Dibbelt, Ben Strasser, and Dorothea Wagner

Karlsruhe Institute of Technology
KIT - ITI Wagner - Box 6980, D-76128 Karlsruhe, Germany
{julian.dibbelt, strasser, dorothea.wagner}@kit.edu

Abstract

We study the problem of computing delay-robust routes in timetable networks. Instead of a single path we compute a decision graph containing all stops and trains/vehicles that might be relevant. Delays are formalized using a stochastic model. We show how to compute a decision graph that minimizes the expected arrival time while bounding the latest arrival time over all sub-paths. Finally we show how the information contained within a decision graph can compactly be represented to the user. We experimentally evaluate our algorithms and show that the running times allow for interactive usage on a realistic train network.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Algorithms, Optimization, Delay-Robustness, Route planning, Public transportation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.1

1 Introduction

In recent years there has been considerable progress in quickly computing optimal journeys in public transportation networks. Unlike road networks, these networks are schedule-based, that is, they (are supposed to) follow a timetable of planned vehicle departures and arrivals. Optimality of journeys is typically based on earliest arrival time subject to other criteria such as number of transfers, latest departure time, or price. See [1] for a recent overview.

In the real-world, however, the scheduled timetable is only worth so much, as train delays occur. Besides prolonging the time spent traveling, delays might make planned transfers to other vehicles impossible. Given todays widespread internet coverage and modern route planning algorithm's flexibility [2, 12, 6] with timetable updates, replanning these missed transfers is not a problem. However, with limited transit service during, e.g., the evening hours, the aggregated delay induced by missed transfers can be considerably more than the original delay. In the worst case, the traveler has to spent the night in the middle of nowhere.

Therefore, it has been proposed to plan journeys already with possible delays in mind [8, 11, 10]. A basic approach might just add sufficiently large buffer times to each transfer. While likely to play out as planned, such journeys would often have unacceptably late arrival times. Obviously, the user would want to also optimize for arrival time and number of transfers, too. One approach to tackle this problem is to compute the set of Pareto-optimal solutions. However, we observe that a single journey is often exclusively either fast or delay-

* Partial support by DFG grant WA654/16-2 and EU grant 288094 (eCOMPASS) and Google Focused Research Award.

resilient but not both. The Pareto set therefore often does not contain a single-path-journey that is good with respect to all criteria simultaneously.

Hence, in this paper, we consider the problem of computing a travel plan (i.e., a collection of journeys) that does not break if delays occur but is still fast overall. To that end, we consider fast journeys but require that for every step of the plan there is always a backup, i.e., a viable alternative towards the target in case of missed transfers. We refer to such a plan as delay-robust. Note that our intuition about delay-robustness goes beyond just avoiding tight transfers. Tight transfers at a frequently serviced station might be unproblematic. Conversely, a fastest (i.e. earliest arrival time) journey is not necessarily part of a good plan, if it transfers at a stop where no good backup departs. Interestingly, our approach also (implicitly) optimizes the number of transfers (as fewer transfers means fewer situations for journeys to break), and if there are several stops at which the user can transfer between two trains, it prefers to transfer at "larger" stations with more connecting trains (giving more options in case something breaks).

We represent the computed plan in the form of a *decision graph* that tells for every transfer how to continue in case of different delays (including no delay). While our primary goal is to compute the travel plan in advance (so that the traveler might print it), please note that this plan can easily be recomputed based on the current delay situation of the network as our query times are well below a second and we do not employ heavy preprocessing.

In [7] we introduced the Connection Scan family of algorithms and very briefly described the core idea of our approach to delay-robust routing. Since then we have extended the approach significantly and present our newer results in this paper. Among the new contributions are techniques to represent the decision graphs compactly and to reduce their size. Our paper is structured as following: In Section 2 we give a brief overview over related work. We then formally define in Section 3 what a timetable is. Using this terminology we describe in Section 4 our delay model. In terms of this model we then define in Section 5 formally what a decision graph and its expected arrival time is. We show some basic properties about the problem of computing a decision graph with minimum expected arrival time and give an optimal solution algorithm. We observe that in practice decision graphs can get large. We therefore propose in Section 6 some strategies to reduce the amount of information presented to the user. Finally we present in Section 7 an experimental evaluation of the proposed algorithms.

2 Related Work

There has been a lot of research in the area of train networks and delays and many of these papers were published at past ATMOS conferences. In contrast to our algorithm most of them compute single paths through the network instead of subgraphs containing all backups. To make this distinction clear we refer to such paths as *single-path-journeys*. The authors of [8] define the reliability of a single-path-journey and propose to optimize it in the Pareto-sense with other criteria such as arrival time or the number of transfers. The availability of backups is not considered. The authors of [5], based on delays occurred in the past, search for a single-path-journey that would have provided close to optimal travel times in every of the observed situations. The authors of [11] propose to first compute a set of safe transfers (i.e. those that always work). They then develop algorithms to compute single-path-journeys that arrive before a given latest arrival time and only use safe transfers or at least minimize use of unsafe transfers. In [10], a robust primary journey is computed such that for every transfer stop a good backup single-path-journey to the target exists. However,

the backups do not have their own backups. The approach optimizes the primary arrival time subject to a maximum backup arrival time. The authors of [9] study the correlation between real world public transit schedules in Rom and compare them against the single-path-journeys computed by state-of-the-art route planners based on the scheduled timetable. They observe a significant discrepancy and conclude that one should consider the availability of good backups already at the planning stage. The authors of [2] examine delay-robustness in a different context: Having computed a set of transfer patters on a scheduled timetable in a urban setting, they show that single-path-journeys based on these patterns are still nearly optimal even when introducing delays. The conclusion is that these sets are fairly robust (i.e., the paths in the delayed timetable often use the same or similar patterns). In [3] the authors propose to present to the user a small set of transfer patterns that cover most optimal journeys. They show that in an urban setting few patterns are enough to cover most single-path-journeys. In a different line of work, the authors of [4] investigate how a delay-perturbed timetable will evolve over time using stochastic methods. Their study shows that this is a computationally expensive task (running time in the seconds) if the delay model accounts many real-world details. Using a model with such a degree of realism therefore seems unfeasible for delay-robust route planning (requiring query times in the milliseconds).

3 Basics

Every random variable X in this work is denoted by capital letters, is continuous, non-negative and has a maximum value $\max X$. We denote by $P[X \leq x]$ the probability that the random variable is below some constant x and by $E[X]$ the expected value of X.

A timetable is a triple $(\mathcal{S}, \mathcal{C}, \mathcal{T})$ of *stops* $\mathcal{S}$, (elementary) *connections* $\mathcal{C}$ and *trips* $\mathcal{T}$. In terms of these we define a set of *rides* $\mathcal{R}$. A stop is a location where one may enter or exit a train. A connection $c \in \mathcal{C}$ is a tuple $(c_{\text{depstop}}, c_{\text{arrstop}}, c_{\text{deptime}}, c_{\text{arrtime}}, c_{\text{trip}}, \mathrm{D}_c)$ representing a train driving from a *departure stop* c_{depstop} to an *arrival stop* c_{arrstop} without intermediate halt. It is scheduled to depart at *departure time* c_{deptime} and to arrive at *arrival time* c_{arrtime}. We require that $c_{\text{depstop}} \neq c_{\text{arrstop}}$ and $c_{\text{deptime}} < c_{\text{arrtime}}$, that is, connections do not form self-loops and have strictly positive duration. If the train is not on time, it arrives with a random non-negative *delay* D_c. For every connection there is a *maximum delay* $\max \mathrm{D}_c$. A train typically operates several connections in succession, forming a *trip*. The unique trip to which c belongs is $c_{\text{trip}} \in \mathcal{T}$. For two successive connections c^1 and c^2 of a trip, we require $c^1_{\text{arrstop}} = c^2_{\text{depstop}}$ and $c^1_{\text{arrtime}} \leq c^2_{\text{deptime}}$. A ride $(c^{\text{enter}}, c^{\text{exit}})$ is an ordered pair of connections (i.e., $c^{\text{enter}}_{\text{deptime}} < c^{\text{exit}}_{\text{deptime}}$) within a trip (i.e., $c^{\text{enter}}_{\text{trip}} = c^{\text{exit}}_{\text{trip}}$). It represents the user taking a train for several stops without exiting at intermediate stops. We denote by $\mathcal{R}$ the set of all rides. Analogous to connections, we define for every $r \in \mathcal{R}$: $r_{\text{depstop}} = c^{\text{enter}}_{\text{depstop}}$, $r_{\text{arrstop}} = c^{\text{exit}}_{\text{arrstop}}$, $r_{\text{deptime}} = c^{\text{enter}}_{\text{deptime}}$, $r_{\text{arrtime}} = c^{\text{exit}}_{\text{arrtime}}$, $r_{\text{trip}} = c^{\text{enter}}_{\text{trip}}$, and $\mathrm{D}_r = \mathrm{D}_{c^{\text{exit}}}$.

A $(\mathrm{s}, \tau, \mathrm{t})$-*journey* is a sequence of rides $r^1 \dots r^n$. We refer to s as the *source stop*, to τ as the *source time* and to t as the *target stop*. For every journey we require that $\forall i : r^i_{\text{arrstop}} = r^{i+1}_{\text{depstop}}$, $\forall i : r^i_{\text{arrtime}} \leq r^{i+1}_{\text{deptime}}$, $\mathrm{s} = r^1_{\text{depstop}}$, $\tau \leq r^1_{\text{deptime}}$ and $\mathrm{t} = r^n_{\text{arrstop}}$. A journey is *safe* if $\forall i : r^i_{\text{arrtime}} + \max \mathrm{D}_{r^i} \leq r^{i+1}_{\text{deptime}}$ is fulfilled, i.e., even when the trains are delayed no transfer can break. Analogous to connections and rides, we define $j_{\text{depstop}} = r^1_{\text{depstop}}$, $j_{\text{deptime}} = r^1_{\text{deptime}}$, $j_{\text{arrstop}} = r^n_{\text{arrstop}}$, $j_{\text{arrtime}} = r^n_{\text{arrtime}}$, and $\mathrm{D}_j = \mathrm{D}_{r^n}$. The $(\mathrm{s}, \tau, \mathrm{t})$-*earliest (safe) arrival* problem consists of finding a (safe) $(\mathrm{s}, \tau, \mathrm{t})$-journey j minimizing j_{arrtime}. We denote by $\mathrm{ea}(\mathrm{s}, \tau, \mathrm{t})$ and $\mathrm{esa}(\mathrm{s}, \tau, \mathrm{t})$, respectively, the arrival time of an optimal (safe) j.

3.1 Modeling Rational

Many publications on public transit networks consider timetables where a finite connection set repeats, e.g., every day. As such these timetables are infinite and *periodic*. In contrast—unless explicitly stated otherwise—we consider *finite* timetables. We choose this modeling as most real world timetables do not repeat perfectly. However, we require that the timetable in our system spans a sufficiently long period to answer all relevant queries. (Experiments show that our approach scales to at least thirty days in a realistic setting.) Note that for finite timetables there is naturally a latest connection.

We do not explicitly use minimum change times at stops but implicitly encode them in D_c. It does not matter whether a train is always delayed by x minutes or whether the user always needs x minutes to walk from platform to platform. Following most recent papers on delay-robust timetable routing we omit footpaths from our model. Note that omitting footpaths in an urban public transit network may be problematic. However this abstraction is perfectly fine in a long-distant train setting, such as the one which we use in our experiments. In Appendix A we sketch a way of incorporating them.

4 Delay Model

A crucial component of any delay-robust routing system is choosing against which types of delays the system should be robust and how to model these delays. This choice has deep implications throughout the whole system. While a too simplistic model does not yield useful routes, a too complicated model makes routing algorithms too inefficient to be feasible in practice. For the model chosen in [11] it is NP-hard to determine whether a transfer is safe or not. Instead, we propose a simplified stochastic model where this is constant time. While our model that does not cover every situation and is not delay-robust in every possible scenario, it works well enough to give useful routes with backups.

The central simplification is that we assume that all random variables are independent. Clearly, in reality this is not always the case. However, if delays between many trains interact then the timetable perturbation must be significant. Train tracks blocked for an extended period of time is a specific example of significant perturbation. As reaction to such a perturbation even trains in the medium or distant future need to be rescheduled (or arrive at least not on-time). The set of possible outcomes and the associated uncertainty is huge. Accounting for every outcome seems infeasible to us. We argue that if the perturbation is large then we can not account for all possible recovery scenarios in advance. Instead, the user should replan his journey based on the actual situation. Furthermore, even if we could account for all scenarios, we would still face the problem of explaining every possible outcome to the user, which is a show-stopper in practice. Our model therefore only accounts for small disturbances as we only intend to be robust against these.

Formally, our model contains one random variable D_c per connection c. This variable indicates with which delay the train will arrive at c_{arrstop}. We assume that all connections depart on time. This assumption does not induce a significant error because it roughly does not matter whether the incoming or the outgoing train is delayed. Furthermore, we assume that every connection c has a maximum delay, i.e., $\max \mathrm{D}_c$ is a finite value. Finally, we assume that all random variables are independent. Delays between trips are independent because if they were not then the perturbation would be large. We can assume that delays within a trip are independent: The typical user would not be willing to exit a trip at a stop just to reenter it later on at a different stop.

The only remaining modeling issue is to define what distribution the random variables D_c should have. An obvious choice is to estimate a distribution based on historic delay data. However, this has two shortcomings: (i) it is hard to get access to delay data (we do not have it), and (ii) you need to have records of many days with precisely the same planned schedule. Suppose for example that the user is in the middle of his journey and a significant perturbation occurs. The operator then adjusts the short-term timetable to reflect this and the user wants to reroutes based on this adjusted data. With historic data this often is not possible because this exact recovery scenario may never have occurred in the past and almost certainly not often enough to extrapolate from the historic data.

For these reasons we propose to use synthetic delay distributions that are only parametrized on the planned timetable. We propose to add to each connection c a synthetic delay variable D_c that depends on the minimum change time m of c_{arrstop} and on a global[1] *maximum delay parameter* d. We define D_c as follows: $\forall x \in (-\infty, 0] : P[\mathrm{D}_c \leq x] = 0$, $\forall x \in (0, m] : P[\mathrm{D}_c \leq x] = \frac{2x}{6m-3x}$, $\forall x \in (m, m+d] : P[\mathrm{D}_c \leq x] = \frac{31(x-m)+2d}{30(x-m)+3d}$, and $\forall x \in (m+d, \infty) : P[\mathrm{D}_c \leq x] = 1$. The function is illustrated in Figure 1 and the rational for our design is given in Appendix B.

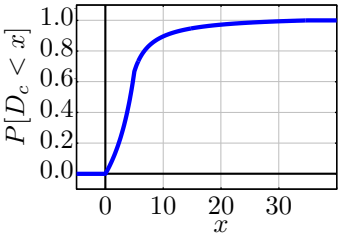


Figure 1 Plot showing $P[\mathrm{D}_c \leq x]$ in function of x for $m = 5$ and $d = 30$.

5 Decision Graphs

In this section, we first formally define the decision graph and then discuss three problem variants: (i) the unbounded, (ii) the bounded, and (iii) the α-bounded MEAT problems. The first two are of more theoretical interest, whereas the third one has the highest practical impact. We prove basic properties of the unbounded and bounded problems and show a relation to the earliest safe arrival problem. Finally, we give an exact optimal-solution algorithm for the unbounded problem on finite networks and show how it is adapted to solve the bounded and the α-bounded problems.

5.1 Formal Definition

A $(\mathrm{s}, \tau, \mathrm{t})$-*decision graph* from source stop s to target stop t with the user departing at time τ is a directed reflexive-loop-free multi-graph $G = (V, A)$ whose vertices correspond to stops (i.e., $V \subseteq \mathcal{S}$) and whose arcs correspond to rides r (i.e., $A \subseteq \mathcal{R}$) directed from r_{depstop} to r_{arrstop}. There may be several rides between a pair of stops, but they must be of part of different trips and depart at different times. We formalize this as: $\forall r^1, r^2 \in A : r^1_{\text{deptime}} \neq r^2_{\text{deptime}} \vee r^1_{\text{depstop}} \neq r^2_{\text{depstop}}$. We require that the user must be able to reach every ride and must always be able to get to the target. Formally, we require that for every $r \in A$ there exists a $(\mathrm{s}, \tau, r_{\text{depstop}})$-journey j with $j_{\text{arrtime}} \leq r_{\text{deptime}}$ to reach the ride, and a safe $(r_{\text{arrstop}}, r_{\text{arrtime}} + \max \mathrm{D}_r, \mathrm{t})$-journey j' to reach the target. To exclude decision graphs with unreachable stops, we require that every stop in V except s and t have non-zero in- and out degree. For simplicity, we further require that $\mathrm{s} \neq \mathrm{t}$.

We first recursively define the *expected arrival time* $e(r)$ (short EAT) of a ride $r \in A$ and define in terms of $e(r)$ the EAT $e(G)$ of the whole decision graph G. If $r_{\text{arrstop}} = t$, we define $e(r) = r_{\text{arrtime}} + E[\mathrm{D}_r]$. Otherwise $e(r)$ is defined in terms of other rides. Denote by

[1] d is global since we lack per-train data. Our approach can be adjusted, if such data became available.

$q_1 \dots q_n$ the sequence of rides ordered by departure time, departing at r_{arrstop} after r_{arrtime}, i.e., every ride that the user could reach after r arrives. Denote by $d_1 \dots d_n$ their departure times and set $d_0 = r_{\text{arrtime}}$. We define $e(r) = \sum_{i \in \{1 \dots n\}} P[d_{i-1} < \mathrm{D}_r < d_i] \cdot e(q_i)$, i.e., the average of the EATs of the connecting rides weighted by the transfer probability. Note that this definition is well-defined because $e(r)$ only depends on $e(q)$ of rides with a later departure time, i.e., $r_{\text{deptime}} < q_{\text{deptime}}$. Further notice that $P[\mathrm{D}_r < d_n] = 1$. Otherwise no safe journey to the target would exist invalidating the decision graph.

We denote by G^{first} the ride $r \in A$ with minimum r_{deptime}. This is the ride that the user must initially take at s. We define the *expected arrival time* $e(G)$ (short EAT) of the decision graph G as $e(G^{\text{first}})$. Furthermore, the *latest arrival time* $G_{\text{maxarrtime}}$ is the maximum $r_{\text{arrtime}} + \max \mathrm{D}_r$ over all $r \in A$. Note that by minimizing $G_{\text{maxarrtime}}$ we can bound the worst case arrival time giving us some control over the arrival time variance.

The *unbounded* $(\mathrm{s}, \tau, \mathrm{t})$*-minimum expected arrival time* (short MEAT) problem consists of computing a $(\mathrm{s}, \tau, \mathrm{t})$-decision graph G minimizing $e(G)$. The bounded $(\mathrm{s}, \tau, \mathrm{t})$-*MEAT* problem consists of computing a $(\mathrm{s}, \tau, \mathrm{t})$-decision graph G minimizing $e(G)$ subject to a minimum $G_{\text{maxarrtime}}$. As a compromise between bounded and unbounded we further define the α-bounded MEAT problem: We require that $G_{\text{maxarrtime}} - \tau \leq \alpha\,(\text{esa}\,(\mathrm{s}, \tau, \mathrm{t}) - \tau)$, i.e., the maximum travel time must not be bigger than α times the delay-free optimum. Notice that the bounded and 1-bounded MEAT problems are equivalent.

5.2 Decision Graph Existence

▶ **Lemma 1.** *There is a* $(\mathrm{s}, \tau, \mathrm{t})$*-decision graph* G *iff there exists a safe* $(\mathrm{s}, \tau, \mathrm{t})$*-journey* j*.*

Proof. By definition there must be a safe $(G^{\text{first}}_{\text{arrstop}}, G^{\text{first}}_{\text{arrtime}} + \max \mathrm{D}_{G^{\text{first}}}, \mathrm{t})$-journey j'. Prefixing j' with G^{first} yields j. Conversely, as the rides in the sequence of j already form a $(\mathrm{s}, \tau, \mathrm{t})$-decision graph we have shown both directions. ◀

A direct consequence of this lemma is that the minimum $G_{\text{maxarrtime}}$ over all $(\mathrm{s}, \tau, \mathrm{t})$-decision graphs G is equal to $\text{esa}(\mathrm{s}, \tau, \mathrm{t})$. Using this observation we can reduce the bounded MEAT problem to the unbounded MEAT problem. Formally stated:

▶ **Lemma 2.** *An optimal solution* G *to the bounded* $(\mathrm{s}, \tau, \mathrm{t})$*-MEAT problem on timetable* T *is an optimal solution to the unbounded* $(\mathrm{s}, \tau, \mathrm{t})$*-MEAT problem on a timetable* T' *where* T' *is obtained by removing all connections* c *with* c_{arrtime} *above the* $\text{esa}(\mathrm{s}, \tau, \mathrm{t})$*.*

Proof. There are two central observations needed for the proof: First, every $(\mathrm{s}, \tau, \mathrm{t})$-decision graph on timetable T' is a $(\mathrm{s}, \tau, \mathrm{t})$-decision graph on the strictly larger timetable T. Second, every safe $(\mathrm{s}, \tau, \mathrm{t})$-journey in T' is an earliest safe $(\mathrm{s}, \tau, \mathrm{t})$-journey in T. Suppose that a $(\mathrm{s}, \tau, \mathrm{t})$-decision graph G' on T' would exist with a suboptimal $G'_{\text{maxarrtime}}$ then there would also exist a safe $(\mathrm{s}, \tau, \mathrm{t})$-journey j' in T' with a suboptimal j'_{arrtime}, which is not possible by construction of T', which is a contradiction. ◀

Having shown how to explicitly bound $G_{\text{maxarrtime}}$ it is natural to ask what would happen if we dropped this bound and solely minimized $e(G)$. For this we consider the infinite timetable T_p illustrated and defined in Figure 2. Notice that T_p is constructed such that it does not matter whether the user arrives at a at moment $1 + 4\mathbb{N}$ or at b at moment $3 + 4\mathbb{N}$ as the two states are completely symmetric with the stops a and b swapping roles. By exploiting this symmetry we can reduce the set of possibly optimal $(\mathrm{s}, 0, \mathrm{t})$-decision graphs to 2 elements: the decision graph G^1 that waits at a and never goes over b, and the decision graph G^2 that oscillates between a and b. The corresponding expected arrival times are

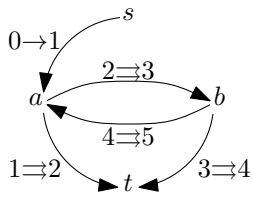


Figure 2 A timetable T_p has 4 stops: s, a, b and t. The arrows denote connections. An arrow annotated with its departure time and arrival time. A simple arrow ($\rightarrow$) denotes a single non-repeating connection. A double arrow ($\rightrightarrows$) is repeated every 4 time units, i.e. $1 \rightrightarrows 2$ is a shorthand for $1 + 4i \rightarrow 2 + 4i$ for every $i \in \mathbb{N}$. All connections are part of their own trip and have the same delay variable D. We define $P[\mathrm{D} = 0] = p$ (with $p \neq 0$) and $P[\mathrm{D} < 1] = 1$.

defined using $e(G^1) = p\,(2 + E[\mathrm{D}]) + (1 - p)\,(7 + E[\mathrm{D}])$ and $e(G^2) = p\,(2 + E[\mathrm{D}]) + (1 - p)\,\big(3 + e(G^2)\big)$. The later equation can be resolved to $e(G^2) = E[\mathrm{D}] - 1 + \frac{3}{p}$. We can solve $e(G^1) < e(G^2)$ in terms of p. The result is that G^1 is better if $p < \frac{\sqrt{43}-4}{9} \approx 0.28$. If they are equal then G^1 and G^2 are equivalent and otherwise G^2 is better.

This has consequences even for timetables with a finite $\mathcal{C}$. One could expect that to compute a decision graph it is sufficient to look at a time-interval proportional to its expected travel time: It seems reasonable that a connection scheduled to occur in ten years would not be relevant for a decision graph departing today with an expected travel time of one hour. However, this intuition is false in the worst case: Consider the finite sub-timetable T' of the periodic timetable T_p that encompasses the first ten years (i.e., we "unroll" T_p for ten years). For p>0.28, an optimal $(\mathrm{s}, 0, \mathrm{t})$-decision graph will use all connections in T', including the ones in ten years (as G^2 would). Fortunately, the bounded MEAT problem does not suffer from this weakness: No connection arriving after $\mathrm{esa}(\mathrm{s}, 0, \mathrm{t})$ can be relevant. Therefore, even on infinite networks the bounded MEAT problem always admits finite solutions.

5.3 Solving the Unbounded MEAT Problem

The unbounded MEAT problem can be solved to optimality on finite networks, and by extension also the α-bounded MEAT problem. Our algorithm is based on the Profile Connection Scan algorithm [7] and exploits three key insights: (i) Every optimal $(\mathrm{s}, \tau, \mathrm{t})$-decision graph $G = (V, A)$ contains for every ride $r \in A$ an optimal $(r_{\mathrm{depstop}}, r_{\mathrm{deptime}}, \mathrm{t})$-decision sub-graph, (ii) exchanging an optimal $(r_{\mathrm{depstop}}, r_{\mathrm{deptime}}, \mathrm{t})$-sub-graph of G with another optimal $(r_{\mathrm{depstop}}, r_{\mathrm{deptime}}, \mathrm{t})$-decision graph yields an optimal $(\mathrm{s}, \tau, \mathrm{t})$-decision graph, and (iii) the first connection of all decision sub-graphs H of G have a later departure time, i.e., $G^{\mathrm{first}}_{\mathrm{deptime}} \leq H^{\mathrm{first}}_{\mathrm{deptime}}$. Together these three ingredients give rise to a dynamic programming algorithm. Denote for every $c \in \mathcal{C}$ an optimal $(c_{\mathrm{depstop}}, c_{\mathrm{deptime}}, \mathrm{t})$-decision graph by $G(c)$ subject to $G(c)^{\mathrm{first}}_{\mathrm{enter}} = c$, i.e., the user must start his travel sitting in c. Further denote by $e(c) = e(G(c))$ the EAT of $G(c)$ if one exists and $e(c) = \infty$ otherwise. Our base algorithm works in two phases: (i) Compute $e(c)$ for all $c \in \mathcal{C}$, (ii) extract a desired $(\mathrm{s}, \tau, \mathrm{t})$-decision graph using the $e(c)$. The actual algorithm used to solve the α-bounded problem variant differs slightly and is detailed in Section 5.4.

5.3.1 Phase 1: Computing all Expected Arrival Times

The core idea consists of starting with the trivial sub-timetable with $\mathcal{C} = \emptyset$ and then iteratively adding the connections ordered decreasing by departure time. When c is inserted

we compute $e(c)$. Once all connections are inserted the phase is finished. During computations we maintain two extra data structures: (i) for every trip $h \in \mathcal{T}$ we store the best known EAT using h, i.e., $q(h) = \min_{c_{\text{trip}}=h} e(c)$, (ii) for every stop $p \in \mathcal{S}$ we store the list of outgoing connections $q(p,1), q(p,2) \dots q(p,n_p)$ ordered increasing by departure time for which $G(q(p,i))$ is an optimal $(q(p,i)_{\text{depstop}}, q(p,i)_{\text{deptime}}, \text{t})$-decision graph (i.e. the connections for which the restriction $G(q(p,i))^{\text{first}}_{\text{enter}} = q(p,i)$ can be dropped). We refer to the lists of (ii) as *profiles*. Observe that an optimal (x,y,z)-decision graph is also an optimal (x,y',z)-decision graph for $y' \leq y$. We therefore know that the profiles must be domination reduced, i.e., $\forall p,i,j : i \leq j \Rightarrow e(q(p,i)) \leq e(q(p,j))$. In terms of these data structures we can describe the actual algorithm: Denote by c the connection being inserted. The data structures are correct for the sub-timetable without c. We need to correct them to accommodate for c. As we insert connections decreasing by departure time we know that c has a minimum c_{deptime} among all connections in the sub-timetable at the moment c is inserted. As additionally $c_{\text{deptime}} < c_{\text{arrtime}}$ must hold, we know that if a decision graph G uses c then $G^{\text{first}}_{\text{enter}} = c$ must hold. The user has three options when c arrives. We calculate the EATs for all three and the minimum is the desired $e(c)$. The options are: (i) remain seated, (ii) arrive at the target stop t, and (iii) arrive at c_{arrstop} and pick another train. The EAT for (i) is just $q(c_{\text{trip}})$. For (ii) the EAT is $c_{\text{arrtime}} + E[\text{D}_c]$ or ∞, depending on whether $\text{t} = c_{\text{arrstop}}$. For (iii) computing the EAT is slightly more complex: If $c_{\text{arrtime}} + \max \text{D}_c > q(c_{\text{arrstop}}, n_p)_{\text{deptime}}$ then no safe journey to t exists and the EAT is ∞. Otherwise the EAT is $\sum_i P[d_{i-1} < \text{D}_c < d_i]\, e(q(c_{\text{arrstop}}, i))$ where d_i is a shorthand for $q(c_{\text{arrstop}}, i)_{\text{deptime}}$and d_0 is c_{arrtime}. After computing $e(c)$ we need to repair the q data structures to accommodate for c: the trips are fixed using $q(c_{\text{trip}}) \leftarrow \min\{e(c), q(c_{\text{trip}})\}$ and we add c to c_{depstop}'s profile if it is not dominated, i.e., if $e(c) < e(q(c_{\text{depstop}}, 1))$. If $e(c) = e(q(c_{\text{depstop}}, 1))$ then we may add it but do not have to. If $c_{\text{deptime}} = q(c_{\text{depstop}}, 1)_{\text{deptime}}$ then the first profile element is replaced and otherwise the profile list grows by one element.

5.3.2 Phase 2: Extracting Decision Graphs

We extract a $(\text{s}, \tau, \text{t})$-decision graph $G = (V, A)$ by enumerating all rides in A. The stop set V can then be inferred from A. At the core, our algorithm uses a min-priority queue that contains connections ordered increasing by their departure time. Initially, we add the earliest connection in the profile of s to the queue. While the queue is not empty we pop the earliest connection c^1 from it. Denote by $c^2 \dots c^n$ all subsequent connections in the trip c^1_{trip}. The desired ride $r = (c^1, c^i)$ is given by the first i such that $e(c^1) \neq e(c^{i+1})$ (or $i = n$ if all are equal). We add r to G. If $c^i_{\text{arrstop}} \neq \text{t}$ we add the following connections to the queue: (i) All connections in the profile of c^i_{arrstop} departing between c^i_{arrtime} and $c^i_{\text{arrtime}} + \max \text{D}_{c^i}$, and (ii) the first connection in the profile of c^i_{arrstop} departing after $c^i_{\text{arrtime}} + \max \text{D}_{c^i}$.

5.3.3 Optimizations

Instead of storing the EAT for each connection we store the values inside of the stop profiles, resulting in better memory locality. We further store the corresponding rides in the profiles to avoid the iteration over the trip's connections during the extraction. Recall that all connections in a decision graph must be reachable. We exploit this by skipping connections c for which $c_{\text{deptime}} < \text{ea}(\text{s}, \tau, c_{\text{depstop}})$ instead of adding them to the network. We determine this earliest arrival time by running a basic one-to-all Connection Scan.

(a) Expanded **(b)** Compact

Figure 3 Decision graph representations from Karlsruhe at 9:00 to Berlin.

5.4 Solving the α-Bounded MEAT Problem

We assume that $\mathcal{C}$ is stored as an array ordered by departure time. To solve the α-bounded $(\mathrm{s}, \tau, \mathrm{t})$-MEAT problem we perform the following steps: (i) run a binary search on $\mathcal{C}$ to determine the earliest connection c^{first} departing after τ, (ii) run a trip-aware one-to-one Connection Scan from s to t that assumes all connections c are delayed by $\max \mathrm{D}_c$ to determine $\mathrm{esa}(\mathrm{s}, \tau, \mathrm{t})$ (iii) let $\tau_{\text{last}} = \tau + \alpha \cdot (\mathrm{esa}(\mathrm{s}, \tau, \mathrm{t}) - \tau)$ and run a second binary search on $\mathcal{C}$ to find the last connection c^{last} departing before τ_{last}, (iv) run a trip-unaware one-to-all Connection Scan from s restricted to the connections from c^{first} to c^{last} to determine all $\mathrm{ea}(\mathrm{s}, \tau, \cdot)$, (v) run Phase 1 of the base algorithm scanning the connections from c^{last} to c^{first} skipping connections c for which $c_{\text{arrtime}} > \tau_{\text{last}}$ or $\mathrm{ea}(\mathrm{s}, \tau, c_{\text{depstop}}) \leq c_{\text{deptime}}$ does not hold, and finally (vi) run Phase 2 of the base algorithm, i.e., extract the $(\mathrm{s}, \tau, \mathrm{t})$-decision graph.

6 Decision Graph Representation

In the previous section we described how to compute decision graphs. In practice this is not enough and we must be able to represent the graph in a form that the user can effectively comprehend. The main obstacle here is to prevent the user from being overwhelmed with information. A secondary obstacle is how to actually layout the graph. In this section we solely focus to reducing the amount of information. Producing actual layouts is still the focus of ongoing research. The presented drawings were created by hand.

6.1 Expanded Decision Graph Representation

The expanded decision graph subdivides each node v into slots $s_{v,1} \dots s_{v,n}$ that correspond to moments in time that an arc arrives or departs at v. The slots in each node are ordered from top to bottom in chronological order. Each arc (u, v) connects the corresponding slots $s_{u,i}$ and $s_{v,j}$. To determine his next train the user has to search for the box corresponding to his current stop and pick the first departure slot after the current moment in time. The arrows guide him to the box corresponding to his next stop. Figure 3a illustrates this.

6.2 Compact Decision Graph Representation

The scheduled arrival time of trains is an information contained in the expanded decision graph that is not strictly necessary. (Besides being inaccurate because of delays.) To decide on the next connecting train to take at a transfer stop, it suffices to know the available next rides departing after "now", that is, the actual arrival time at that stop.

The compact decision graph exploits this observation by removing the arrival time information from the representation. Each arc (u, v) connects the corresponding departure slot $s_{u,i}$ directly to the stop v instead of a slot. Time slots that only appear as arrival

slots are removed. If two outgoing arcs of a node u have the same destination and depart subsequently (as for high-frequency lines), they are grouped and only displayed once. The compact decision graph is never larger than the expanded one and most of the time significantly smaller. See Figure 3b for an example.

6.3 Relaxed Dominance

Decision graphs exist that contain rides that have near to no impact on the EAT. Removing them increases the EAT by only a small amount, resulting in an almost optimal decision graph that can be significantly smaller. To exploit this, we introduce a *relaxation tuning parameter* β. EATs are regarded as equal if their difference is below β. We only add a connection c to the profile q if $e(c) \leq e(q(c_{\text{depstop}}, 1)) - \beta$.

6.4 Displaying only the Relevant Subgraphs

In many scenarios we have a canvas of fixed size. If even the compact relaxed decision graph is too large to fit, we can only draw parts of it. We observe that the decision graph extraction phase does not rely on the actual distributions of the delay variables D_c but only on $\max D_c$. It extracts all connections departing in a certain interval I, plus the first connection directly afterwards. Reducing the size of I reduces the number of rides displayed, while still guaranteeing that backup rides exist (they just are not displayed). We refer to the size of I as *display window*. Given an upper bound γ on the number of arcs in the compact (or expanded) representation, we use a binary search to determine the maximum display window and draw the corresponding subgraph. (Note that in the worst case the display window has size 0. Then the decision graph degenerates to a single-path-journey.)

7 Experiments

For our experiments we used on a single core of a Xeon E5-2670 at 2.6 GHz, with 64 GiB of DDR3-1600 RAM, 20 MiB of L3 and 256 KiB of L2 cache and we used g++ 4.7.1 with -O3.

Table 2 Instance Size.

#Stop	16 991
#Conn.	55 930 920
#Trip	3 965 040

The timetable is based on the data of `bahn.de` during winter 2011/2012. We extracted every vehicle except for most buses[2] as we mainly target train networks. We focus on long-distance networks where delays have a significantly larger impact than in high-frequent inner-city transit. We removed footpaths longer than 10 min, connected stops with a distance below 100 m, and then contracted stops connected through footpaths adjusting their minimum change times resulting in an instance without footpaths. We pick the largest strongly connected component to make sure that there always exists a journey (assuming enough days are considered). We extract one day of maximum operation (i.e. extract everything regardless of the day of operation and remove exact duplicates). We then replicated this day 30 times to have a timetable spanning about one month of operation. The detailed sizes are in Table 2. We ran 10 000 random queries. Source and target stop are picked uniformly at random. The source time is chosen within the first 24h. We filter queries out that have an minimum delay-free travel time above 24h.

Our experimental results are presented in Table 1. The compact representation is smaller by a factor of 2 in terms of arcs than the expanded one. As expected, a larger relaxation

[2] Not having buses explains the significant instance size difference compared to [12].

Table 1 The time (in ms) needed to compute a decision graph and its size. Arcs is the number of arcs in the compact representation. The number of rides corresponds to the number of arcs in the expanded representation. The maximum delay parameter is set to 1h. We report average, maximum and the 33%-, 66%- and 95%-quantiles.

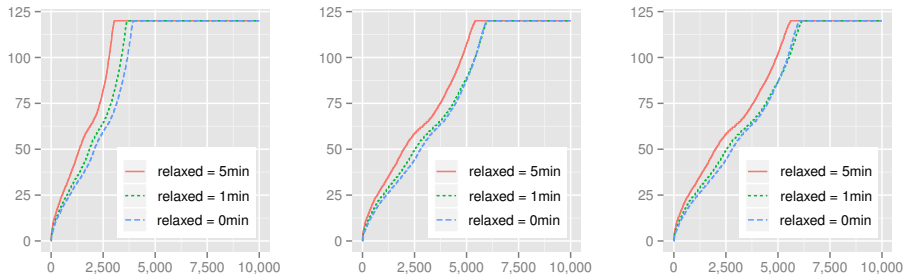
		Unbounded				2.0-Bounded				1.0-Bounded			
		Time	Stops	Rides	Arcs	Time	Stops	Rides	Arcs	Time	Stops	Rides	Arcs
0min-Relax	Avg	6 452	12	98	42	138	12	87	35	26	9	45	19
	33%	6 209	7	22	10	84	7	22	10	16	7	15	7
	66%	7 407	13	70	31	162	13	69	31	27	10	40	19
	95%	7 635	25	349	125	312	24	330	119	66	19	149	57
	Max	7 805	280	35 450	28 848	817	173	5 540	4 703	288	38	1 607	366
1min-Relax	Avg	5 122	12	88	39	116	12	73	31	25	9	39	17
	33%	4 628	8	26	12	75	8	25	12	16	6	14	7
	66%	6 026	13	66	31	136	13	64	30	26	10	36	17
	95%	6 368	24	284	110	249	24	257	100	64	18	123	52
	Max	6 595	50	12 603	6 558	685	50	1 576	478	240	37	1 390	289
5min-Relax	Avg	4 180	11	66	33	100	11	51	25	24	9	29	15
	33%	3 845	8	24	12	66	8	23	11	15	6	13	6
	66%	4 808	13	53	26	115	12	51	25	25	10	30	15
	95%	5 028	22	178	82	216	22	155	74	61	17	84	42
	Max	5 159	54	6 640	3 220	553	54	760	285	196	34	590	183

parameter gives smaller graphs. Increasing the α-bound leads to larger graphs and running times grow. The running times of unbounded queries are proportional to the timespan of the timetable (i.e. 30 days). On the other hand, the running times of bounded queries depend only on the maximum travel time of the journey. This explains the gap in running time of two orders of magnitude. As the maximum values are significantly higher than the 95%-quantile we can conclude that the graphs are in most cases of manageable size with a few outlines that distort the average values. Upon closer inspection we discover that most outliers with large decision graphs connect remote rural areas, where even no "good" delay-free journey exists. We can therefore not expect to find any form of robust travel plan.

In Figure 4 we evaluate the display window such that the extracted graphs have less than 25 arcs in the compact representation. Recall that this modifies what is displayed to the user. It is still guaranteed that backups exist. As the 1.0-bounded graphs are smaller than 2.0-bounded graphs we can display more, explaining the larger display window. The difference between 2.0-bounded graphs and unbounded graphs is small. A greater relaxation parameter also reduces the graph size and thus allows for slightly larger display windows. If there is no "good" way to travel the decision graphs degenerate to single-path-journeys.

8 Conclusion & Future Work

We studied variants of the MEAT-problem to compute decision graphs. Experimentally, we determined that, while the resulting graphs are not tiny, they are sufficiently small to be useful to the user. Running times are small enough to allow interactive usage. Possible direction for future work include: (i) incorporate trains that wait on other trains, (ii) explore the feasibility of stochastic footpaths (note that Appendix A discusses deterministic footpaths),



(a) 1.0-Bounded **(b)** 2.0-Bounded **(c)** Unbounded

Figure 4 Display windows in min (y-axis) for each of the 10 000 test queries (x-axis) ordered increasingly. The maximum delay parameter is set to 2h.

and (iii) determine whether more sophisticated delay models can be solved efficiently.

Acknowledgment. We would like to thank Thomas Pajor for his valuable input.

9 Bibliography

References

1 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller–Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. Technical Report MSR-TR-2014-4, Microsoft Research, 2014.

2 Hannah Bast, Jonas Sternisko, and Sabine Storandt. Delay-robustness of transfer patterns in public transportation route planning. In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'13)*, OpenAccess Series in Informatics (OASIcs), pages 42–54, September 2013.

3 Hannah Bast and Sabine Storandt. Flow-based guidebook routing. In *Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX'14)*, pages 155–165. SIAM, 2014.

4 Annabell Berger, Andreas Gebhardt, Matthias Müller–Hannemann, and Martin Ostrowski. Stochastic delay prediction in large train networks. In *Proceedings of the 11th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'11)*, volume 20 of *OpenAccess Series in Informatics (OASIcs)*, pages 100–111, 2011.

5 Kateřina Böhmová, Matúš Mihalák, Tobias Pröger, Rastislav Šrámek, and Peter Widmayer. Robust routing in urban public transportation: How to find reliable journeys based on past observations. In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'13)*, OpenAccess Series in Informatics (OASIcs), pages 27–41, September 2013.

6 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-based public transit routing. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 130–140. SIAM, 2012.

7 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2013.

8 Yann Disser, Matthias Müller–Hannemann, and Mathias Schnee. Multi-criteria shortest paths in time-dependent train networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, June 2008.

9 Donatella Firmani, Giuseppe F. Italiano, Luigi Laura, and Federico Santaroni. Is timetabling routing always reliable for public transport? In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'13)*, OpenAccess Series in Informatics (OASIcs), pages 15–26, September 2013.

10 Marc Goerigk, Sascha Heße, Matthias Müller–Hannemann, and Marie Schmidt. Recoverable robust timetable information. In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'13)*, OpenAccess Series in Informatics (OASIcs), pages 1–14, September 2013.

11 Marc Goerigk, Martin Knoth, Matthias Müller–Hannemann, Marie Schmidt, and Anita Schöbel. The price of robustness in timetable information. In *Proceedings of the 11th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'11)*, volume 20 of *OpenAccess Series in Informatics (OASIcs)*, pages 76–87, 2011.

12 Ben Strasser and Dorothea Wagner. Connection scan accelerated. In *Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX'14)*, pages 125–137. SIAM, 2014.

A Footpaths

In the paper we omitted footpaths from the model as nearly all related work on delay-robust routing does so and because the timetable used in the experiments is still meaningful without. Incorporating footpaths is not as straight-forward as it seems at first. The main obstacle is finding a meaningful formalization. Depending on this formalization solving the problem can be easy or hard from an algorithmic point of view.

In [7], we used a very simplistic model with the following assumptions: (i) footpaths are always exact and never delayed, (ii) the user can use a footpath right after he exits a train, and (iii) the user can use a footpath at the start and end of his journey. This model implies that if the user walks from a stop p to a stop q and misses the train he wanted to get, then he will wait at q for his next train and not try to walk back to p.

Assuming this model, our algorithm can be extended to incorporate footpaths as following: Every time we add a non-dominated connection c with corresponding EAT τ to the profile of stop p, we iterate over all footpaths from a stop q to p with duration d. We add c also to q if $\tau - d$ is not dominated at q. This covers initial and intermediate footpaths. Final footpaths need special attention. We incorporate them by maintaining an array A that maps every stop ID onto the footpath distance to the target stop. Case ii in Phase 1 of the algorithm, where the user arrives at the target, must be modified. We do not check whether the current connection c arrives at the target stop t but we look up in A the distance from c_{arrstop} to t.

B Rational For Synthetic Delays

It is important to realize that there are many different ways to come up with formulas for synthetic delays. The lack of any effectively accessible ground truth makes any conclusive experimental evaluation of their quality very difficult. The only real criteria that we have is "intuitively reasonable". The approach presented here is by no means the final answer to

the question of what is the best synthetic delay distribution. In this section we describe the rational for our design decisions.

We define for every connection c its delay D_c by defining its cumulative distribution function $f_{m,d}(x)$, where d is the maximum delay of c and m the minimum change time at c_{arrstop}. Our delays do not depend on any other parameters than m and d. We have the following hard requirements on $f_{m,d}$ resulting from our algorithm:

- $f_{m,d}(x)$ is a probability, i.e., $\forall x : 0 \leq f(x) \leq 1$
- $f_{m,d}(x)$ is a cumulative distribution function and therefore non-decreasing, i.e., $\forall x : f'_{m,d}(x) \geq 0$
- $\max \mathrm{D}_c$ should be $m + d$, i.e., $\forall x \geq m + d : f(x) = 1$
- Our model does not allow for trains that arrive too early, i.e.,$\forall x < 0 : f(x) = 0$

These requirements already completely define what happens outside of $x \in (0, m + d)$. Because of the limitations of current hardware, we have two additional more fuzzy but important requirements:

- We need to evaluate $f_{m,d}(x)$ many times. The formula must therefore not be computationally expensive.
- Our algorithm computes a lot of $(f_{m,d}(x_1) + a_1) \cdot (f_{m,d}(x_2) + a_2) \cdot (f_{m,d}(x_3) + a_3) \cdots$ chains. The chain length reflects the number of rides in the longest journey considered during the computations. As 64-bit-floating points only have a limited precision we must make sure that order of magnitude of the various values of $f_{m,d}$ do not differ too much. If they do differ a lot then the less likely journeys have no impact on the overall EAT because their impact is rounded away.

Finally there are a couple of soft constraints coming from our intuition:

- $f(m)$ is the probability that everything works as scheduled without the slightest delay. In practice this does happen and therefore this should have reasonable high probability. On the other hand a too high $f(m)$ can lead to problems with rounding. We set $f(m) = \frac{2}{3}$ as we believe that it is a good compromise.
- We want f to be continuous.
- The maximum variation should be at $x = m$, i.e., $f'(m)$ should be the unique local maximum of f'.
- Initially the function should grow slowly and then once $x = m$ is reached the growth should slow down. This can be formalized as $f''(x) > 0$ for $x \in (0, m)$ and $f''(x) < 0$ for $x \in (m, m + d)$.

We define f using two piece function f_1 and f_2. For these pieces we assume $m = 5\text{min}$ and $d = 30\text{min}$ and scale them to accommodate for different values, as following:

$$f_{m,d}(x) = \begin{cases} 0 & \text{if } x < 0 \\ f_1(\frac{5x}{m}) & \text{if } 0 \leq x \leq m \\ f_2\left(\frac{30(x-m)}{d}\right) & \text{if } m < x < m + d \\ 1 & \text{if } m + d \leq x \end{cases}$$

It remains to define f_1 and f_2. We started with a $-1/x$ function and shifted and stretched the function graphs until we ended up with something that looks "intuitively reasonable".

$$f_1(x) = \frac{2x}{3(10 - x)}$$
$$f_2(x) = \frac{31x + 60}{30(x + 3)}$$

The resulting function f fulfills all requirements and is illustrated in Figure 1.